Case for Usability: Designing Outdoor Augmented Reality Games

by

Tiffany Wang

B.S., Computer Science, 2007
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

August 2008

Author: ______________________________

Department of Electrical Engineering and Computer Science
August 22, 2008

Certified by: ______________________________

Eric Klopfer
Associate Professor, Department of Urban Studies and Planning
Thesis Supervisor

Accepted by: ______________________________

Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Case for Usability: Designing Outdoor Augmented Reality Games

by

Tiffany Wang

Submitted to the
Department of Electrical Engineering and Computer Science

August 22, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Creating a successful Outdoor Augmented Reality (OAR) game can be a complicated process. With every new feature added to the OAR toolset, games gain more levels of complexity, grow in size of content, and become increasingly difficult to produce and manage. In order to identify plausible methods to help alleviate some of the difficulties when creating OAR games, a heuristic usability evaluation of the existing Game Editor toolkit and an assessment of the needs of game designers were made as part of this research. Two new applications, the Desktop Editor and Remote Editor, were designed, prototyped, and evaluated by new and experienced game designers. The Desktop Editor offers new methods of visualizing and working with data which have proven to be useful features for creating games but also add difficulties to overall learnability. The Remote Editor offers on-location game editing capabilities which help expedite many of the tasks involved with creating and testing OAR games. Feedback and user tests suggest that the new applications offer valuable ideas for game editing features that would be beneficial in future iterations of the OAR Game Editor toolkit.

Thesis Supervisor: Eric Klopfer
Title: Associate Propfessor, Department of Urban Studies and Planning

Acknowledgements

I would like to acknowledge all those who have helped contribute to the completion of this project.

Many thanks to my project manager, Judith Perry, for her leadership and guidance since my first joining the OAR project and for her patience and support throughout this challenging endeavor. Thanks to Professor Eric Klopfer for the opportunity to be a part of the Teacher Education Program and the OAR team, for his advice on various aspects of the project, and for the constant supply of diverse blends of coffee. Thanks to my fellow developer, Lisa Stump, for sharing her technical knowledge and experience, for always challenging my decisions to find the best possible solution, and for stepping up on the project while I finished my research. Thanks to RJ Silk for doing an amazing job refactoring the backend code base and making the process of developing new editing tools so much easier.

Thanks to the following project collaborators for their feedback throughout the project and for being such great resources of knowledge and experience with the OAR game development process: Marleigh Norton and Josh Sheldon at MIT; Chris Holden, Mingfong Jan, John Martin, and James Mathews at the University of Wisconsin Madison; Patrick O'Shea at Harvard University; and Matt Dunleavy at Radford University.

Thanks to Professor Rob Miller for introducing me to the field of User Interface research and design, for the privilege of being his teaching assistant this past semester, and for providing me with a solid foundation for undertaking this project.

Most importantly, endless thanks to my friends and family for all their support and encouragement and constant reminders to enjoy life outside of lab and get enough sleep.

Table of Contents

1. Introduction

There has been a recent outbreak of mysterious illnesses around the MIT campus. Though no causes have yet been determined, suspicions have prompted investigations of construction sites and reactor labs in the area. You have been commissioned by the MIT administration to lead the efforts in uncovering the source of the epidemic and devise a plan for remediation. The only tool provided for the job: a handheld device with location tracking capabilities. Good luck!

The story may be made up, but the scenario is real. Middle school students at a recent science fair were given this same introduction and sent out into the MIT campus equipped with handheld computers and GPS devices. This is just one example of an Outdoor Augmented Reality game, a new learning tool which can be customized to any class curriculum and is designed to help students learn through immersion with the world around them.

This chapter presents the background information and context for this project. The first section provides a definition of Augmented Reality and the purpose of the technology. The second section discusses the applications of Augmented Reality, focusing on location-based educational games, also known as Outdoor Augmented Reality games. Lastly, the third section describes the creation of these types of Augmented Reality games.

1.1 Augmented Reality

Augmented Reality (AR) is a form of technology which enhances reality by combining the digital world with the real world; intangible information with the tangible. Unlike Virtual Reality, which fully immerses the user in an artificial environment, Augmented Reality still allows users to interact directly with their surroundings and instead "augments" the environment with digital information in the form of audio clips, sensory cues, or visual overlays.

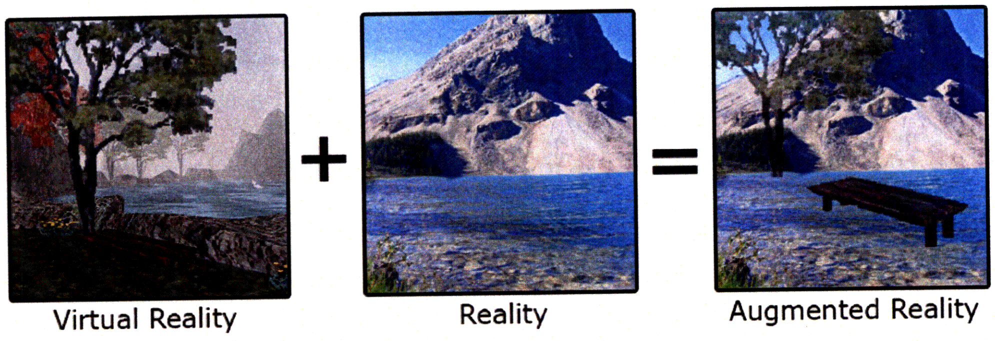


Figure 1.1 Conceptual diagram of Augmented Reality. Augmented Reality is a combination of virtual reality and actual reality and belongs to a class of technology known as "Mixed Reality" (Milgram & Kishino, 1994).

By combining virtual reality with actual reality, AR attempts to transcend the both. The auxiliary information superimposed onto the user's environment can enhance the user's perception and interactions with the world around them. In turn, the environment can provide the information with a real world context which helps the information be more readily conveyed to the user. Imagine learning about the people and events involved in the Battle of Bunker Hill while exploring the actual site of the battle as opposed to reading about the battle in a school library hundreds of miles away. Ultimately, the goal of developing AR systems is to provide the user with an overall more engaging interactive experience and to enable visualization and understanding of information that would not be possible in a purely real or virtual world.

1.2 Outdoor AR Games

AR systems have been applied to a variety of fields including architecture, medicine, military training, and entertainment. This project involves the application of Augmented Reality to games for education, specifically the *creation* of Outdoor AR games.

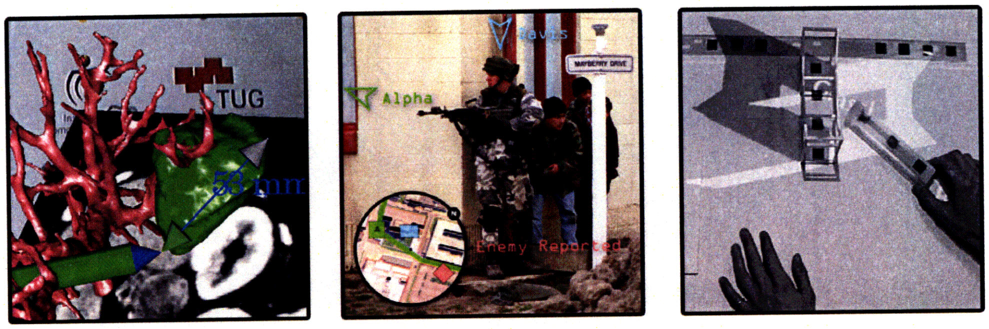


Figure 1.2 Augmented Reality applications. Augmented Reality systems are currently used in a variety of applications including heart surgery, military training games, and architectural planning.

Outdoor AR (OAR) games are games played at an on-site location using handheld devices with GPS (Global Positioning System) tracking capabilities. Conceptually, OAR games can be thought of as a mix between role-playing games and scavenger hunts: players assume a character role and must complete tasks and solve problems by gathering information from their surroundings. Unlike scavenger hunts which rely on physical clues, OAR games utilize virtual clues in the form of virtual game objects placed at tagged locations throughout the environment. These game objects are virtual in the sense that they are visible and accessible only through the handheld screen. Utilizing the GPS device, players can track their location as they move throughout the playing field and can "visit" virtual game objects by walking into the object's hotspot.

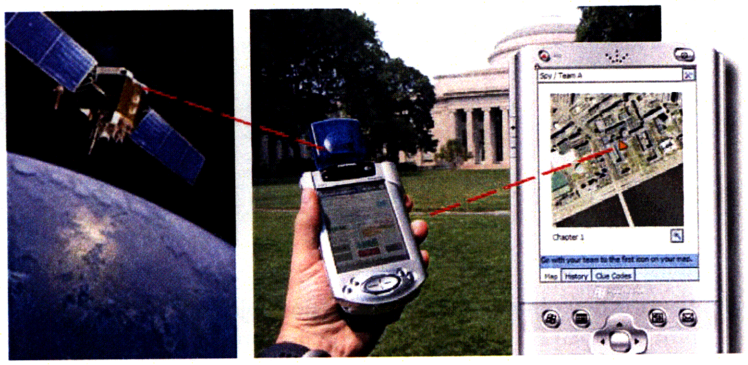


Figure 1.3 Location tracking in Outdoor AR games. Outdoor AR games are played at a specific outdoor location. Global coordinates read from a GPS device are translated into game coordinates used to position the player on the screen.

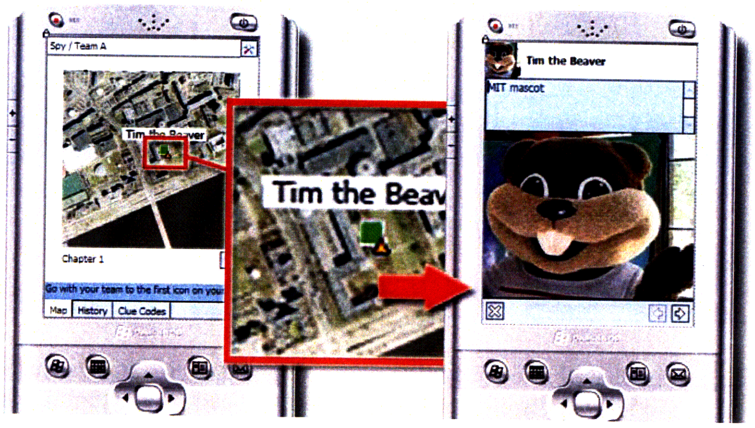
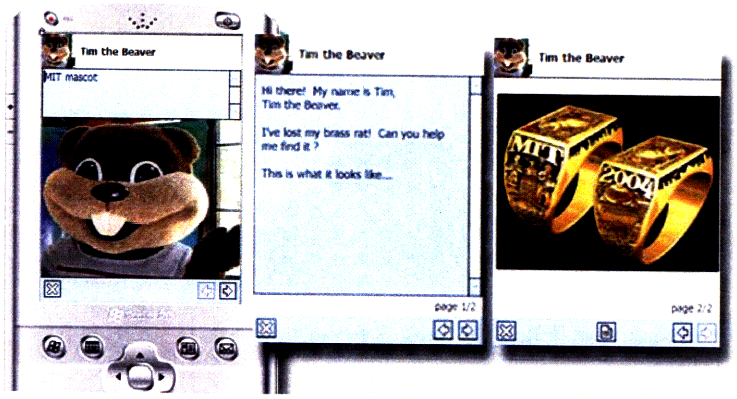


Figure 1.4 Interacting with virtual game objects. Virtual game objects are visible and accessible only through the handheld screen. A player can "visit" a game object by walking into the object's hotspot.

Game objects can take on the form of NPCs (non-playable characters), items, and even buildings. Each game object has a unique name, picture, and description and may provide a player with information by displaying dialog text or media files such as images, audio and video clips, and web page articles on the handheld screen.



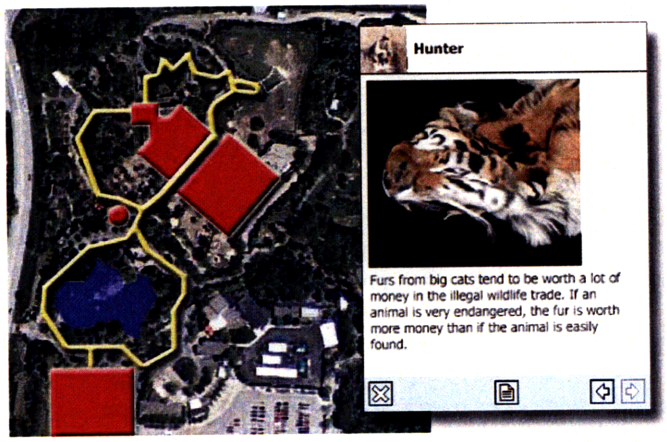
Hi there! My name is Tim, Tim the Beaver.

I've lost my brass rat! Can you help me find it ?

This is what it looks like...

Figure 1.5 Game object information. Each game object has a unique profile and can provide a player with information in the form of dialog text or media files.

One example of an OAR game is "Zoo Scene Investigators", a game developed in 2007 as part of a collaboration between the MIT Teacher Education Program (MIT TEP) and the Columbus Zoo and Aquarium (Norton, 2007). "Zoo Scene Investigators" is played at the zoo in Columbus, Ohio and is designed to teach students about the illegal wildlife trade. The premise of the game is that an attempted robbery has occurred at the zoo and zoo officials are trying to decide whether the man in custody is guilty of trying to steal confiscated animal artifacts. Game participants may choose to play as a police officer, a secret agent, or a spy and must piece together clues gathered from virtual characters located throughout the zoo in order to reach a final conclusion.



Furs from big cats tend to be worth a lot of money in the illegal wildlife trade. If an animal is very endangered, the fur is worth more money than if the animal is easily found.

Figure 1.6 Zoo Scene Investigators. "Zoo Scene Investigators" is an Outdoor AR game about the illegal wildlife trade played at the Columbus Zoo and Aquarium in Columbus, Ohio.

OAR games have been used to teach a variety of other subjects and are one of the newest tools being used in (or more accurately, *outside* of) classrooms to help students develop math, science, and literacy skills and become more engaged with school curriculum. "Environmental Detectives" (Klopfer & Squire, 2005), the first OAR game developed by MIT TEP, is an environmental mystery game in which students work together to uncover the source of a toxic spill and propose a solution for cleaning up the contaminants. Research from that particular implementation identified OAR games as an effective tool for teaching students about environmental engineering and scientific investigation. The majority of the students was excited by the technology and enjoyed the immersive style of learning.

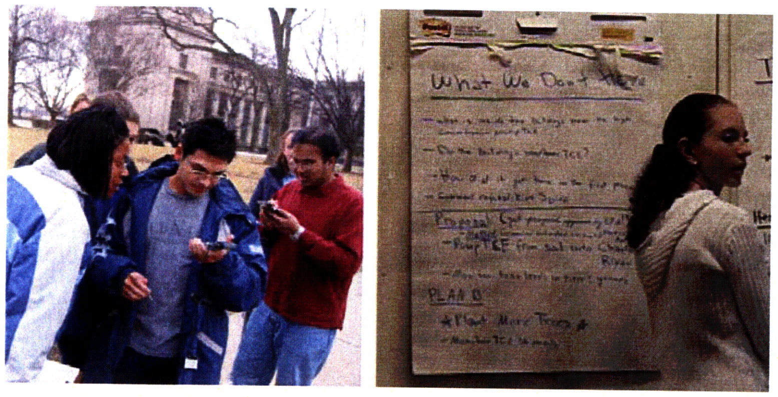


Figure 1.7 Outdoor AR games and learning. Outdoor AR games are now being piloted in schools in hopes of helping students better develop math, science, and literacy skills and become more engaged with class curriculum.

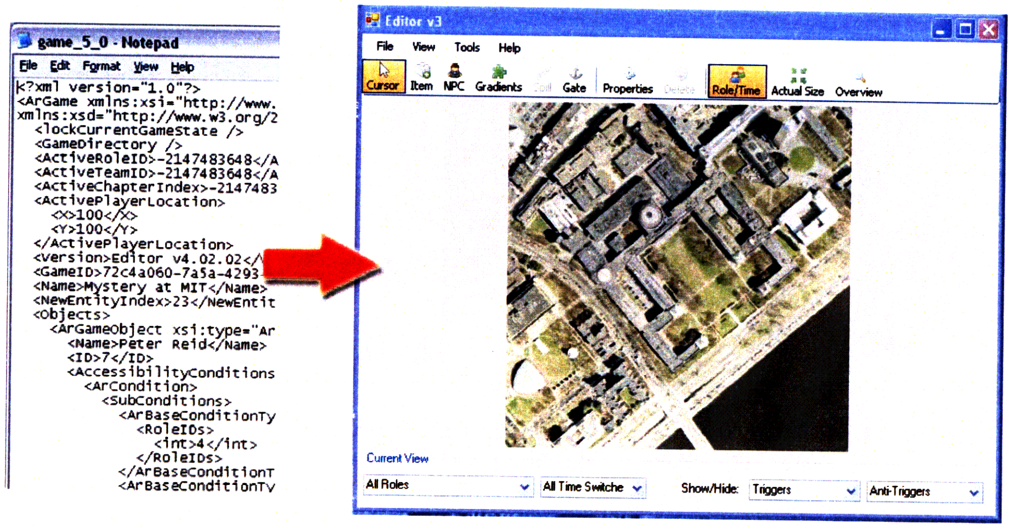
OAR games are only one instantiation of outdoor location based games. Similar technologies have been developed by a number of research groups for both educational and entertainment purposes. One example is Mediascapes, developed by HP (Hewlett Packard) Labs in collaboration with the Futurelab, U.K. Like OAR games, Mediascapes are developed for GPS-enabled handheld devices and utilize digital overlays to tag geographic locations with audio, video, and other forms of media. Mediascapes do not require any structured storyline and have been used by the public for a variety of applications including guided walks, interactive tours, and games (Stenton, 2007). Games Atelier developed at the Waag Society, Amsterdam, is another project dedicated to games in learning using location-based games (Veelo, 2008). Students use mobile phones to add and collect information to and from the environment according to predefined game templates, then share their information with one another over an online network. Similar to MIT TEP, the most recent research surrounding the Games Atelier project has been focused on allowing students to create their own location-based education games.

Figure 1.8 Related AR games research. One other example of outdoor location based games is Mediascapes, developed by the HP labs in collaboration with the Futurelab, U.K.

1.3 Creating Outdoor AR Games

OAR games are played by loading a game file into the Outdoor AR Game Engine application installed on handheld devices. The game file is specific to each particular game implementation and contains all the game settings and content including map and coordinates, game object profiles, dialog text, and media files.

OAR game files were first created by meticulously hand-generating XML (Extensible Markup Language) files in a standard plain-text editor. Since game designers were not always familiar with the XML language, the task of populating game data usually fell to the OAR developers. Needless to say, this was a very tedious and error-prone process during production and testing. With the development of the Outdoor AR Game Editor by MIT TEP, a toolkit with a graphical user interface became available to game designers for creating custom OAR games. The Game Editor is a desktop application that allows designers to create an OAR game for any location with any theme and any content by simply uploading a map and its coordinates, populating the game with game objects, and deploying the game file onto a handheld device to play on site. By abstracting away underlying details of the OAR game structure, designers now have more control over game content which enhances the overall quality of games.



```
<?xml version="1.0"?>
<ArGame xmlns:xsi="http://www.
xmlns:xsd="http://www.w3.org/2
  <lockCurrentGameState />
  <GameDirectory />
  <ActiveRoleID>-2147483648</A
  <ActiveTeamID>-2147483648</A
  <ActiveChapterIndex>-2147483
  <ActivePlayerLocation>
    <X>100</X>
    <Y>100</Y>
  </ActivePlayerLocation>
  <Version>Editor v4.02.02</
  <GameID>72c4a060-7a5a-4293
  <Name>Mystery at MIT</Name>
  <NewEntityIndex>23</NewEntit
  <Objects>
    <ArGameObject xsi:type="Ar
      <Name>Peter Reid</Name>
      <ID>7</ID>
      <AccessibilityConditions
        <ArCondition>
          <SubConditions>
            <ArBaseConditionTy
              <RoleIDs>
                <int>4</int>
              </RoleIDs>
            </ArBaseConditionT
            <ArBaseConditionTy
```

Figure 1.9 From XML to GUI. The Outdoor AR Game Editor provides a means of creating custom OAR games using a desktop application and abstracts away the need to manually generate game content in XML.

Creating a game may seem simple enough using the existing toolkit, but in reality the creation of a cohesive and effective OAR game is a much more involved process than just dropping game objects onto a map and typing in some text. Perhaps the most important aspect of OAR games is the connection between the game and the physical location. A truly immersive OAR game implementation is more than a linear walk-a-path through the game space, it is a dynamic experience that complements and adapts to the surrounding environment. This level of player engagement requires a thorough understanding of the game's physical location, something that can only be achieved through dedicated exploration and research of the game space.

Furthermore, the existing toolkit offers an extensive set of tools and options for game creation, each adding its own dimension of complexity to the overall game. Every game object has a profile and content which may be specific to player role and game chapter. Game objects have the ability to contain other game objects, to change their visibility based on time, and to change the visibility of other game objects after being "visited" by the player. Team scripts can be applied to override initial location and visibility settings. As the laundry list of features continues to grow, so does the difficulty in managing the complexity of games.

The current challenge and the focus of this project is to streamline the process of creating OAR games, focusing on the needs of the most common users: game designers.

2. Problem Statement

Creating OAR games can be a very complex process because of the sheer size the scope of a game can take on. Managing all the details of a game becomes increasingly difficult as the amount of game content grows larger and as more features and levels of customization are added to the game. Although the existing Game Editor is a drastic improvement over plain-text editors in terms of managing this complex process, the toolkit is still an XML builder in spirit and does not always provide the ease of use and robustness of features that game designers would like when creating games.

This project is a re-evaluation and extension on the interface of the existing Game Editor toolkit. By focusing on key user interface design principles, investigating the needs of game designers throughout the design and implementation process, and exploring new methods of visualizing and working with game data, this project strives to develop a more "designer-friendly" Game Editor and to remedy many of the interface problems in the existing toolkit in order to improve overall usability.

What is ***usability***? Usability.gov defines usability as "how well users can learn and use a product to achieve their goals and how satisfied they are with that process" (US DHHS). Usability guru, Jakob Nielsen, defines usability as a "quality attribute that assesses how easy user interfaces are to use" (Nielsen, 2003). In short, there is no strict definition. Usability can be defined using a variety of dimensions depending on the application. However, there *does* exist a traditional set of measurements used to evaluate usability. These measurements are ***learnability, memorability, efficiency, errors,*** and ***satisfaction*** (ISO 13407, 1999).

This chapter describes each of the usability measurements in more detail and identifies areas of improvement that can be made to the existing Game Editor interface. Information pertaining to the pros and cons of the Game Editor was gathered from performing a ***heuristic evaluation*** (evaluation based on usability guidelines) of the interface, as well as from surveying OAR game designers

2.1 Learnability and Memorability

Learnability refers to how easily new users accomplish basic tasks the first time they encounter an interface. ***Memorability*** refers to how easily returning users accomplish tasks each *subsequent* time they encounter an interface. Improving learnability and memorability essentially involves making interfaces easier for new users to learn and easier for returning users to remember.

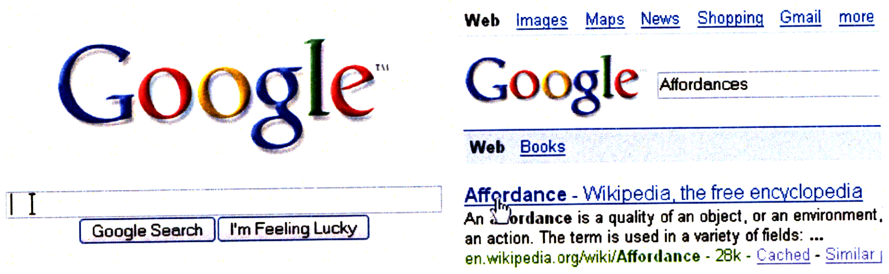
Issues with learnability and memorability in the Game Editor:

- Lack of documentation:

  Documentation of features and instructions for operating the Game Editor are not always readily available since they reside as documents separate from the toolkit. Although it would be ideal for an application to be used without documentation, creating OAR games is not an entirely intuitive process and having an easily accessible resource to reference for help would be useful.

- Insufficient interface cues:

  Cues are nonverbal indications of system functionality. One example of a cue is the ***affordance***, or mapping between the "perceived and actual properties", of controls (Norman, 1988). For example, button controls have an affordance of "click here" while textbox controls with a blinking cursor have an affordance of "type here". Other types of cues include the positioning and visibility of controls and feedback in response to user actions.



Affordance - Wikipedia, the free encyclopedia
An affordance is a quality of an object, or an environment, an action. The term is used in a variety of fields: ...
en.wikipedia.org/wiki/Affordance - 28k - Cached - Similar

Figure 2.1 Cues. Cues are contextual implications of an element's functionality. Some well known cues are cursor changes: an I-bar indicates an area for typing and a pointing hand indicates a clickable hyperlink.

There are a handful of features in the Game Editor, such as Substance Types, Spill Objects, and Team Scripts, which could benefit from additional cues to provide clearer indication of how they should be used. Even experienced game designers are unaware of certain game features or struggle with using more ambiguous parts of the Game Editor.

- Inconsistent interface:

By the *Law of Least Astonishment* (also known as the *Principle of Least Surprise*), an interface should always respond to the user in the way that astonishes him the least (James, 1987). Similar things should look and behave in similar ways. Conversely, different things should be visibly different.

The interface of the Editor has inconsistencies both internally and externally. Externally, the overall layout of the interface differs from applications that are more commonly used by game designers. Internally, the placement, organization, and labeling of similar options may differ from object to object and from dialog form to dialog form. Finding a particular option is not always clear or consistent and many options such as the admin code and GPS settings are grouped haphazardly under the "Miscellaneous" tab in the Game Properties dialog form.

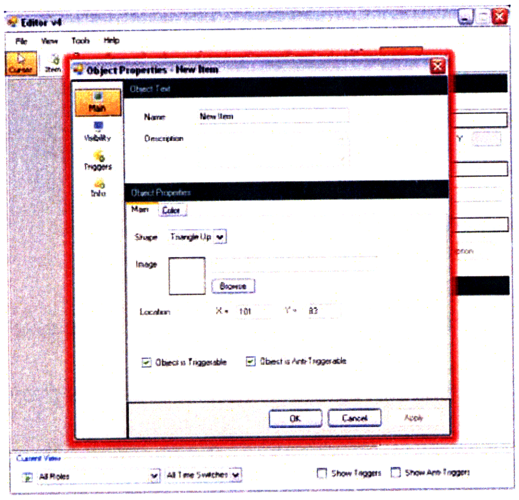
2.2 Efficiency

Efficiency refers to how quickly users can accomplish tasks once they have become familiar with an interface. There are many different methods of improving efficiency depending on the type of interface. More common methods include setting keyboard shortcuts, providing default user inputs, keeping a history of recently performed actions, and allowing customization of interface controls. Although methods of improving efficiency are often catered towards experienced or "power" users, an interface design should strive to improve efficiency for users of all levels.

Issues with efficiency in the Game Editor:

- Restrictive modal-dialog interface:

  With the exception of placing and moving game objects around the game map, all Game Editor interactions such as viewing and editing game object properties are made through ***modal dialog boxes*** that are displayed above the rest of the interface. Modal dialog boxes are dialog boxes that prevent the user from interacting with the application that the dialog was displayed from. As a result, viewing and changing parts of the game can only be done one part at a time. This format may be suitable for a system with a small amount of dynamic data or when working with isolated parts of the game, but turns designing games with a lot of complex, connected parts into a very tedious and often frustrating process.



Name New Item
Description
Shape Triangle Up
Image
Location X = 101 Y = 83
[✓] Object is Triggerable [✓] Object is Anti-Triggerable
All Roles
All Time Switches
[ ] Show Triggers [ ] Show Anti-Triggers

Figure 2.2 Modal dialogs in the Game Editor toolkit. Game object properties are viewed and edited through modal dialog boxes that cover up and prevent interactions with the rest of the interface.

- Lack of functionality shortcuts:

The Game Editor interface does not provide any convenience controls such as keyboard shortcuts, relying solely on mouse input for even simple tasks like saving a game. Although the use of convenience controls has a bit of a learning curve, it can greatly improve efficiency once mastered. The Game Editor *does* provide shortcuts in the form of context menus, but the menus do not always include a complete set of functions.

- Lack of default user input:

Creating an OAR game involves a good deal of data entry such as specifying different properties for each game object. When adding a new game object, game designers are presented with a blank form and required to fill in fields (which may or may not be pertinent to their particular game implementation) before the object can be successfully added. If the designer forgets to set certain properties, it can often lead to unexpected behavior during deployment. For example, if the visibility of a game object is left unspecified, it is defaulted to invisible and the game object will not appear in game. Providing meaningful default values not only improves efficiency, but can help designers better learn the interface by providing examples of proper field entries.

- Non-mobile application:

The Game Editor cannot easily be used remotely because it is a desktop application. Creating a location-based game using a non-mobile tool thus proves to be a prolonged process since not all tasks can be completed directly in the Game Editor. Tasks such as incorporating information from the game site and verifying the placement of game objects on the map must be completed on-site. Without the ability to add content or make corrections on the fly, game designers often need to make multiple trips to the game site before the game is set to their satisfaction.

2.3 Errors

The measurement of ***errors*** is based on the frequency, severity, and ease of recovery from errors made by users. It would be nearly impossible to stop all user errors from ever happening, thus the goal of an interface should be to prevent the occurrence of errors as much as possible and, at the very least, to provide clear error messages.

Common errors made when using the Game Editor:

- Forgetting game content:

Game designers often conceptualize and assemble game content outside the Game Editor and have to manually port (copy-and-paste) the information into the Game Editor. Repetitive and tedious tasks such as these are prone to ***slip errors***, or failures of execution (Reason, 1990). Designers will often make the mistake of copying information into the wrong places or forget to specify information all together. Without having the ability to holistically view all the game content and spot check for holes and mistakes, game designers must methodically verify that all content is present and accounted for. The verification step can be just as repetitive and tedious as porting the content, so designers may not catch all the errors.

- Forgetting game flow settings:

Similar to making errors when populating game content, game designers will often make mistakes when specifying game flow settings. Errors include setting a game object's visibility property to the wrong value, forgetting to set triggers for game objects, and placing the wrong game object within a container object. These types of errors are even harder to detect because of the dependency between multiple game objects, each of which can only be viewed independently of the others as a result of the modal-dialog driven interface.

- Formatting text and images incorrectly:

Since editing the game and playing the game are done on different devices, a disparity can exist between how text and images appear in the Game Editor and how they appear in the Game Engine. To ensure a clean game implementation, designers must ensure that content properties such as image resolution, text length, and web page layout appear as designed on the actual handheld device. Currently, the only way of verifying the look and feel of a game is deploying the game file and all its associated media files to the handheld device and manually playing through the game. The time it takes to test every image for every game object after every modification can quickly add up.

2.4 Satisfaction

Satisfaction refers to how much a user enjoys using the interface. Since it is very subjective measurement that is often loosely defined, satisfaction is difficult to evaluate concretely. There has been a lot of research conducted around developing a structured method of measuring user satisfaction, but none of the proposed instruments have been accepted in the general case. One such instrument is Bailey and Pearson's *39-Factor Computer User Satisfaction*. Users are asked to rate an interface on multiple point scales according to a list of factors such as accuracy, reliability, timeliness, and confidence in the system (Baily & Pearson, 1983).

The satisfaction that a user derives from using an interface is inevitably tied with all the measurements previously mentioned. Not many people would enjoy using an interface that came packaged with a 1,000 page manual, required stepping through a dozen screens to specify a single setting, and crashed sporadically without so much as a warning message. By focusing on ease of use, efficiency of tasks, and an error-free system, improving satisfaction can be easily accomplished for the general user population.

The interface revisions proposed by this project focus primarily on these measurements of usability and try to remedy each of the specific problems outlined above. Since the application being developed is a toolkit for game designers creating OAR games, the revisions also focus on requirements governed by the needs of game designers and the structure of the games. This user and requirements analysis is presented in the next chapter.

3. Requirements Research & Analysis

This chapter details the requirements gathering step of this project. In order to design a new Game Editor interface which accurately reflects the needs of game designers and supports all features of OAR games, it is necessary to first identify what these needs and features are. A crucial step in designing interfaces with good usability is conducting user and task analyses in order to determine how the system can best meet the needs of its users (Hackos & Redish, 1997)

The first section presents the ***user analysis***. The purpose of the user analysis is simple enough: "Know thy user for he is not thyself" (Rubinstein & Hersh, 1984). Since developers are typically not the targeted users, they must find out who the users actually are. Learning about the users consists not only of their individual characteristics, but also situational information such as the environment and social context in which they use the interface. Does the user work in an environment that has a lot of distractions? Does the user work individually or as part of a team?

The second section presents the ***task analysis***. The purpose of the task analysis is to identify the high-level goals and supporting tasks that must be accomplished when using the interface, essentially the purpose of using the interface in the first place. Task analyses can also provide insight into how users typically plan and approach these goals and tasks. The information for the user and tasks analyses was gathered from interviews and surveys conducted with OAR game designers. In total, six game designers with varying degrees of experience creating OAR games participated in the user and task analyses. Questions posed in the interviews and surveys can be found in Appendix A.

The third section presents the ***requirements analysis***. The purpose of the requirements analysis is to create a description of the system's necessary functionality. This information was gathered from reviewing documentation for the existing Game Engine and Game Editor applications.

The last section introduces the proposed approach for designing the new Game Editor interface. The approach was derived from a holistic review of the problems identified in the previous chapter and the information gathered from the following requirements research.

3.1 User Analysis

The targeted users of the Game Editor toolkit are game designers; typically education researchers who are investigating the feasibility and value of using OAR games in a classroom setting. Game designers can be divided into two groups: (1) new designers and (2) experienced designers.

New designers have never created or played an OAR game and are thus unfamiliar with the features and structure of the games. New designers require more support getting started with using the Game Editor and rely more heavily on documentation, application help cues, and guidance from more experienced designers to learn how to use different features of the toolkit. Having never used previous iterations of the Game Editor, new designers do not have any expectations based on earlier iterations of the toolkit.

Experienced designers have medium to high familiarity with creating and playing OAR games. Experienced designers sometimes require a small amount of support with new or previously unused features, but are more concerned with accurately and efficiently revising previously implemented games or creating new games of similar structure. Having used previous iterations of the Game Editor, some experienced designers prefer consistency with earlier interfaces.

Game designers generally have the following characteristics:

- Computer literate.

Designers have experience working on various computer platforms and operating systems. Since the Game Editor toolkit is only supported on Windows PCs, designers typically use desktop computers that are fitted with Windows XP when creating OAR games.

- Experienced with other editing applications.

While creating OAR games, designers use various editing applications including Microsoft Office Suite for editing text and presentations, Adobe Creative Suite for editing image and multimedia, and Macromedia Dreamweaver for editing web pages. Designers are thus more familiar with these types of application interfaces.

- Collaborate with other designers.

Designers work in teams that have anywhere from 1 to 20 people while working on OAR games. Tasks divided up amongst the team include curriculum development, game content creation and formatting, and game building and testing. Since all the tasks are interrelated, designers must share ideas and discuss game content frequently.

- Design and implement games in a very iterative manner.

Creating an OAR game is a very iterative process. Curriculum and content must be reviewed and revised often to ensure they meet education standards and provide a compelling enough game for students. Depending on the life cycle of the project, designers may have to perform major game revisions anywhere from 1 to 5 times, usually in very quick succession.

3.2 Task Analysis

Creating an OAR game is no trivial task. From first inception to final deployment, there are many steps which must be done and redone in order to produce a clean implementation. The game creation process can be broken down into four high-level tasks: (1) design the game, (2) build the game, (3) test the game, and (4) revise the game. Each task is presented below along with more detailed subtasks.

1. Design the game.

- Identify requirements:

  OAR games are created with the purpose of teaching students and therefore must meet certain educational criteria. From the very start, game designers hold interviews and discussions with educators and project facilitators to identify education standards, learning goals, and characteristics of the targeted student age groups. Logistical requirements, such as time constraints, group sizes, levels of supervision, and number of available staff, are also identified in these meetings.

- Scout the location:

  When an OAR game is conceptualized, there is typically a location already in mind. The location must still be evaluated to ensure its suitability for the curriculum and to identify any additional logistical limitations such as out-of-bound areas or places of potential traffic bottlenecks. Game designers may even begin gathering game content and staging portions of the game around landmark objects while on site.

- Gather background information:

Since game designers are typically not actual teachers, they must learn more about the material being presented in order to incorporate proper information into the game. This step may involve sifting through school curriculum plans, reading relevant literature on the subject, or even trying games with similar themes and structure.

- Conceptualize the game:

With all specifications and background information in mind, the game designer must now convert the gathered information into a cohesive game. Content balance is crucial to ensure sufficient information is provided to complete in-game tasks, all roles are contributing equally to the group, and the game meets all educational requirements but is still interesting to play.

2. *Build the game.*

- Prepare game assets:

One of the most compelling aspects of OAR games is the ability to present media content. Typical assets used in games include images, audio and video clips, text articles, and web pages. All these resources must be gathered, correctly formatted, and reviewed by the design team before being placed in-game.

- Assemble the game file:

In order to have a playable OAR game, a game file must be created using the Game Editor toolkit. Information specific to each game implementation falls into two categories: game settings and game content.

Game settings are information related to the logistics of game play. Game settings include the game map image and coordinates, names of the player roles, time and date, and visual settings such as the color of the player icon. ***Game content*** is all the information that was scripted during the design phase. Game content includes game object profile details, text and media data that game objects present to players, as well as other information that describes how game objects are related to one another and how players interact with game objects. A comprehensive list of all the game settings and game content options can be found in the next section.

3. *Test the game.*

- Content testing:

With so much information embedded within an OAR game, missing and erroneous content is a likely and frequent occurrence and must be identified and corrected before the final implementation. As mentioned in the *Errors* section of the *Problem Statement* chapter, common mistakes include forgetting to place files in the proper game directory, associating text and information with the wrong game object, and incorrectly setting the visibility or triggering of game objects.

- On-site testing:

The final implementation of OAR games is played outdoors, so naturally the playability of the game must be tested on-site. Game specifications that must be verified and adjusted include the positioning of the game objects (in relation to landmarks and other game objects) and the ease with which players can "visit" game objects when moving about the playing field.

4. *Revise the game.*

Game properties and content must be continually revised to fix any errors found during the verification phase and also to make adjustments after curriculum reviews with educators and project facilitators. Although any aspect of the game may have to be revised, the majority of revisions involve minor changes such as game object positioning, game object text content, and media file formatting. This task is often repeated multiple times before the final implementation is reached.

3.3 Requirements Analysis

The following tables are a compilation of all game properties, game object properties, and editor functionality that are currently supported or need to be supported in the current Game Editor.

Table 3.1 Game Settings.

Game Settings		
Game Options		
	Game Title	title of OAR game
	Roles	selectable player roles
	Teams	selectable player groups
	Chapters	time based game sections
	Date & Time	start/end time during game
Map Options		
	Map Image & Coordinates	game map and corner coordinates
	Zoom Map & Ratio	optional zoom map and zoom ratio
	Autozoom Boundary	optional autozoom trigger region
	Paint-a-Path Map	optional player-to-path mapping
	Hotspot Region	size of object "visit" region
	Nudge Region	size of object tap to "visit" region
	Player Icon	look and feel of player icon (shape/color)
	Trigger Highlighting	look and feel of trigger indicator (shape/size)
Admin Options		
	Manual Movement	enable/disable thumbpad navigation
	IR Transfers	enable/disable IR beaming
	GPS Settings	GPS setup settings
	Admin Code	access code for in-game admin options

Table 3.2 Game Content.

Game Content		
Role/Chapter/Team Info		
	Introduction	info shown prior to start of game
	Chapter Text	info shown at chapter transitions
	Start/End Tasks	task reminders at start/end of game
	Team Scripts	team-specific settings for object placement/visibility
Point Objects (base properties)		
	Type	classification type
	Location	location on map
	Name	object profile alias
	Image	object profile picture
	Description	object profile description
	Map Icon	look and feel of map icon (shape/color)
	Visibility	conditions for visibility (role/chapter)
	Triggers/Anti-Triggers	objects triggered/anti-triggered by this object
Point Objects (Items/NPCs)		
	Info Pages	information given to player (text/files)
	Documents	media documents given to player
	Task	task message displayed after "visiting" object
Point Objects (Gates)		
	Contained Objects	objects this object "contains"
	Object Codes	associated codes with contained objects
Point Objects (Checkpoints)		
	Code	code needed to clear checkpoint
Substance Objects (Substance Types)		
	Name	substance type alias
	Color	substance type color indicator
	Sampling Time	time to take sample of this type (sec)
	Accessibility	conditions for accessing samples (roles)
	Test Methods	methods used to take samples of this type
Substance Objects (Test Methods)		
	Name	test method alias
	Processing Time	time to process sample using this method (sec)
	Percent Error	possible error in test result
	Accessibility	conditions for accessing method (roles)
Substance Objects (Spill Instances)		
	Name	spill instance alias
	Locations	location of spill foci points
	Substance Type	spill instance substance type
	Drop-off	linear/exponential drop-off rate from center
	Intensity	max spill sample value
	Radius	width of spill instance region
	Visibility	conditions for accessing samples (roles)

Table 3.3 Editor Functions.

Editor Functions	
File Functions	
New Game	create a new game
Load Game	load a previously created game
Save Game	save current game
Recent Files	keep list of recently edited games
Edit Functions	
Add/Edit/Delete	add/edit/delete game objects
Copy/Cut/Paste	copy/cut/paste game objects
Undo/Redo	undo/redo user actions
View Functions	
Map Zoom	zoom in/out on game map
Preview Zoom Map	preview zoom map image
Preview Paint-a-Path	preview paint-a-path map overlay
Filter Game Object Display by:	
- role/chapter	show/hide objects only visible for role/chapter
- point objects	show/hide point objects
- spill objects	show/hide spill gradients
- triggers	show/hide trigger/anti-trigger indicators
Tool Functions	
Map Capture Tool	tool used to obtain game map and coordinates
Check Files	check for missing file paths
Repair Files	repair broken file paths
Package & Deploy Game	Consolidate game files and deploy to handheld

3.4 Project Proposal

Although the design stage is perhaps the most time and resource intensive stage of the game creation process, the majority (if not all) of the tasks must be completed conceptually, outside the scope of the toolkit. Much in the same way that an image must be conceptualized before it can be composed in a graphics editor, an OAR game must be planned and conceptualized before being assembled in the Game Editor. Thus the new design of the Game Editor does not attempt to integrate all game creation stages under a single toolkit, but primarily focuses on helping game designers better manage the complexity of games by alleviating the more tedious and error-prone steps during the building, testing, and revising stages of the creation process. Providing game designer with better editing tools and better control over game data allows them to spend less time building and testing implementations and more time focusing on content and the overall quality of their games.

After reviewing the problems identified by OAR game designers and the information gathered from the requirements research, there was an obvious need for two different interfaces for the next OAR Game Editor. The first interface is a desktop application which extends from the current Game Editor and is intended to be used when working with the majority of the game content. The second interface is a new lightweight mobile counterpart for the handheld device which is intended to be used for on-site planning and game adjustments.

The following chapters describe the design and implementation phases of developing each of the new Game Editors. Design goals, prototype sketches, and implementation details are presented under each section. The ***Desktop Editor*** (desktop application) is introduced in Chapter 4 and the ***Remote Editor*** (handheld application) is introduced in Chapter 5.

4. Desktop Editor

As aptly named, the Desktop Editor is the desktop application of the new OAR Game Editor toolkit. The more complex and heavy-weight of the two toolkit applications, the Desktop Editor's intended use is to handle the majority of the work in composing an OAR game, particularly more complicated tasks such as populating game content and specifying game flow details.

This chapter documents the design and implementation process of creating the Desktop Editor. Development goals, prototype screenshots, and implementation details are presented in the following sections.

4.1 Development Goals

The current Game Editor has already undergone three design iterations and has been an effective tool in creating OAR games. Thus the intention when developing the Desktop Editor was not to create a completely new interface but to improve and build upon the core infrastructure already established by the existing Game Editor. Many elements such as the game map-centric, drag-and-drop method of manipulating game objects were kept intact and re-evaluated to identify key areas for improvement.

In addition to a clean-up of the current Game Editor, several new interface elements and features were added as part of the Desktop Editor's improvements. These features focused on addressing limitations of the current Game Editor, in particular the lack of structure and direction when creating games (learnability and memorability) and the difficulty and tediousness of completing tasks when creating games (efficiency and error).

4.2 Prototypes & Design

This section contains prototype sketches and design descriptions for the most significant additions and modifications made to the Game Editor, organized according to feature.

- Startup Screen and Wizard

The startup screen is the first screen the game designer is presented with upon launching the Desktop Editor. This sort of welcome dialog is a common feature in many other applications and provides the Desktop Editor with a clear entry point from which to navigate. The designer has the option of starting a new game, loading a saved game, or quickly returning to a recently edited game file. These options are not new features but are now directly presented to the designer instead of being tucked away in the File menu.

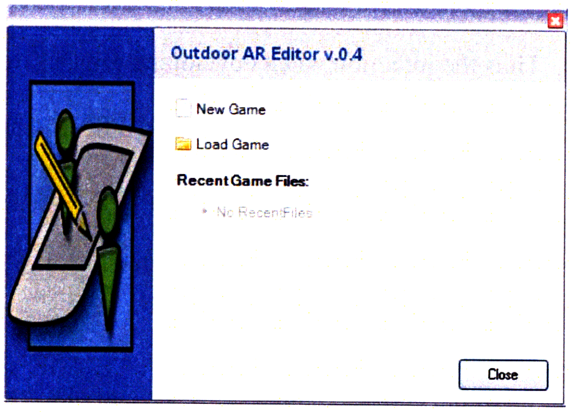


Figure 4.1 Startup Screen Prototype. The startup screen is the welcome screen of the toolkit and is displayed when the Desktop Editor is first launched.

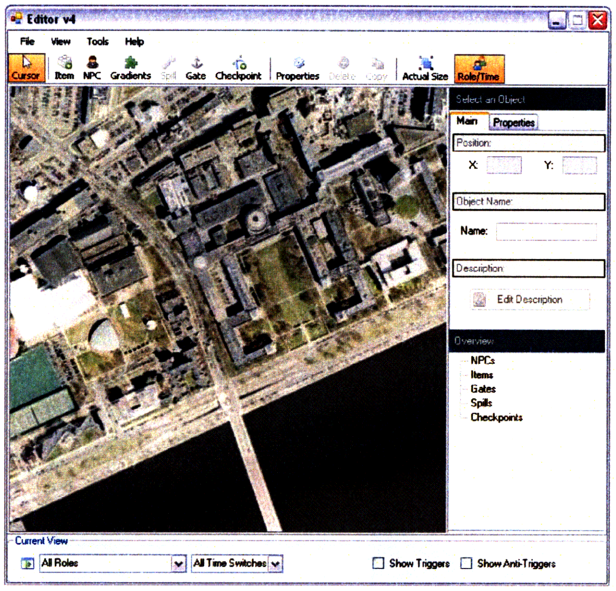


Figure 4.2 Original Game Editor Opening Screen. The current Game Editor launches directly on to the main interface without providing any guidance to the designer.

Electing to start a new game advances the startup screen to a "New Game Setup" wizard. The wizard helps guide the designer through the initial setup of a game. One question considered during design was how much content the game setup wizard would cover: from as much as the A to Z's of creating an OAR game to as little as just the title of the game. In order to help the designer get started as quickly as possible, it was decided to forgo a full tutorial and cover only the bare necessities of setting up a game: title, roles, chapters, and game map.

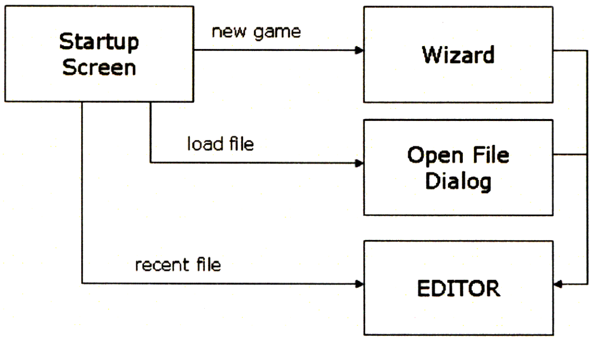
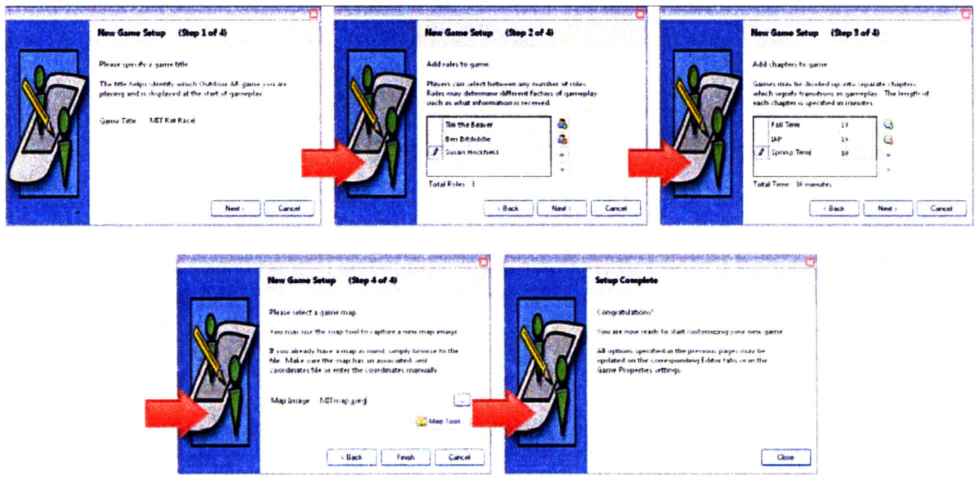


Figure 4.3 Desktop Editor Startup Sequence. The startup screen and wizard provide a step by step guide for the designer through the initial setup of a new game.

In the final design, each of the setup options in the wizard is separated into its own individual screen. By increasing the number of steps through the setup sequence, the wizard avoids overwhelming the designer with too much information at once and has the opportunity to give a brief description and emphasize the functionality of each option; something that would be especially beneficial to a new game designer.


New Game Setup (Step 1 of 4)

Please specify a game title

The title helps identify which Outdoor AR game you are playing and is displayed at the start of gameplay

Game Title: MIT Rat Race

New Game Setup (Step 2 of 4)

Add roles to game.

Players can select between any number of roles. Roles may determine different factors of gameplay such as what information is received.

Tim the Beaver
Ben Bitdiddle
Susan Hockfield

Total Roles: 3

New Game Setup (Step 3 of 4)

Add chapters to game

Games may be divided up into separate chapters which signify transitions in gameplay. The length of each chapter is specified in minutes.

Fall Term	10
IAP	10
Spring Term	10

Total Time: 30 minutes

New Game Setup (Step 4 of 4)

Please select a game map

You may use the map tool to capture a new map image.

If you already have a map in mind, simply browse to the file. Make sure the map has an associated .xml coordinates file or enter the coordinates manually.

Map Image: MITmap.jpeg

Setup Complete

Congratulations!

You are now ready to start customizing your new game.

All options specified in the previous pages may be updated on the corresponding Editor tabs or in the Game Properties settings.

Figure 4.4 Wizard Paper Final Design. The wizard was separated into individual screens to emphasize the purpose of each setup option.

- New Interface Organization

A recurring theme in the Desktop Editor interface is modularity. All editable options in the Desktop Editor are grouped within three main tabs according to related game functionality: "Map", "Object Info", and "Game Info". The "Map" tab is very similar to the main interface of the current Game Editor and encompasses options related to visual properties of game objects such as map placement, visibility conditions, triggers, and icon look-and-feel. The "Object Info" tab is used for editing all game object related content including dialog text and media files, object containment specifications, and access codes. The last tab is used for editing general game content that is player role specific. Content such as introduction text, chapter transition messages, and start and ending task reminders are set up under the "Game Info" tab.

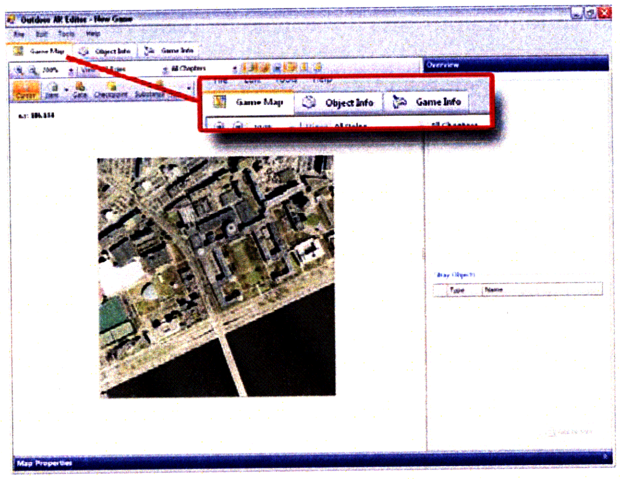


Figure 4.5 New Interface Organization Paper Prototype and Final Design. The Desktop Editor interface is partitioned into three main tabs according to related functionality: map, game object content, and general game content.

The game content within the "Object Info" and "Game Info" tabs are further grouped into pages. Each game object type (Item, NPC, Gate, Checkpoint, and Substance type) and each game info type (Introduction, Chapter Messages, and Start and End Tasks) has its own tabbed page under the "Object Info" and "Game Info" tabs, respectively. Content on each tab and page is kept in sync with all others in order to maintain consistent data. Thus modifications such as changing an object's name on the "Object Info" tab will be properly reflected on the "Map" tab and all other referenced locations in the toolkit.

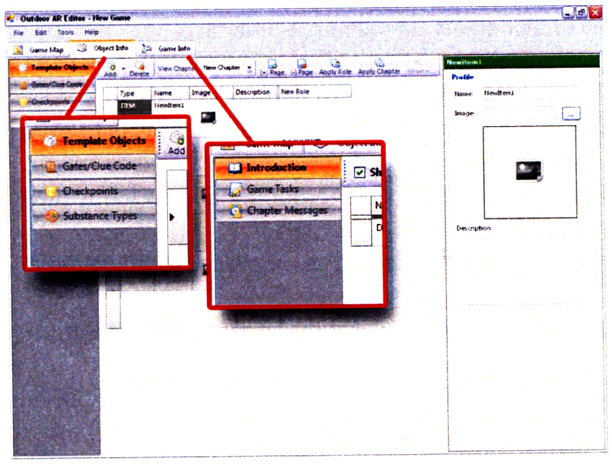


Figure 4.6 Layered Page Tabs. The main tabs are further grouped into pages according to the specific type of game object or game info type.

The organization of editor options was influenced by observations of how game designers typically approached creating OAR games and the sequence in which implementation steps were completed. Regardless of project, the necessary parts identified by game designers for assembling an OAR game included game object placement, content population, and general game setup. The three tabbed pages on the main interface of the Desktop Editor were thus chosen to reflect these major parts. Since different designers have different preferences for the order in which to complete these parts, there arose the question of what would be the best ordering of the tabs. The tabs were originally arranged from general to specific options ("Game Info", "Object Info", "Map") but the order was later reversed to reflect the ordering which game designers generally use to planned out game implementations and also to launch the toolkit onto a familiar interface for the more experienced game designers.

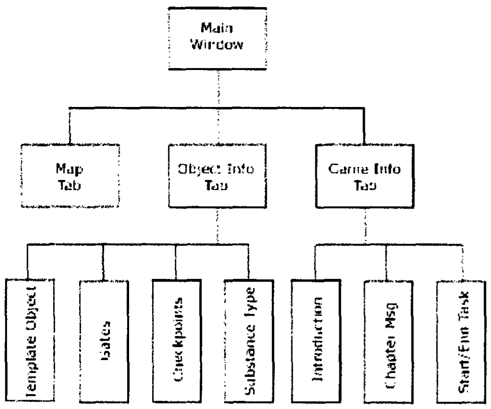


Figure 4.7 New Interface Hierarchy Map. The Desktop Editor interface upholds an theme of modularity throughout its interface.

The main motivation behind the interface reorganization was to give more structure to building games when working with the Game Editor. The grouping allows game designers to plan, build, and test the game in more logical and manageable parts. For example, some designers who preferred to first plan and populate game content found it a bit awkward that they were required to place each game object before being able to start populating the game object's content. The grouping of editor options allows these designers to focus and work on content for all objects in one sweep, then go back and properly position the objects and adjust look and feel when they are ready to work on object placement.

- Content Tables

The content tables are the biggest addition to the Desktop Editor interface. All game object and general game content such as text dialog and media files and chapter transition messages are displayed in an Excel-like grid table layout. All tables are organized with player role column headers and content type row headers, though the exact layout of information varies with the type of content being displayed. For example, the chapter transition messages only take up single rows for each chapter message string while objects such as Items and NPCs may take up multiple rows since each object can have a variable number of dialog pages as well as associated documents and tasks.

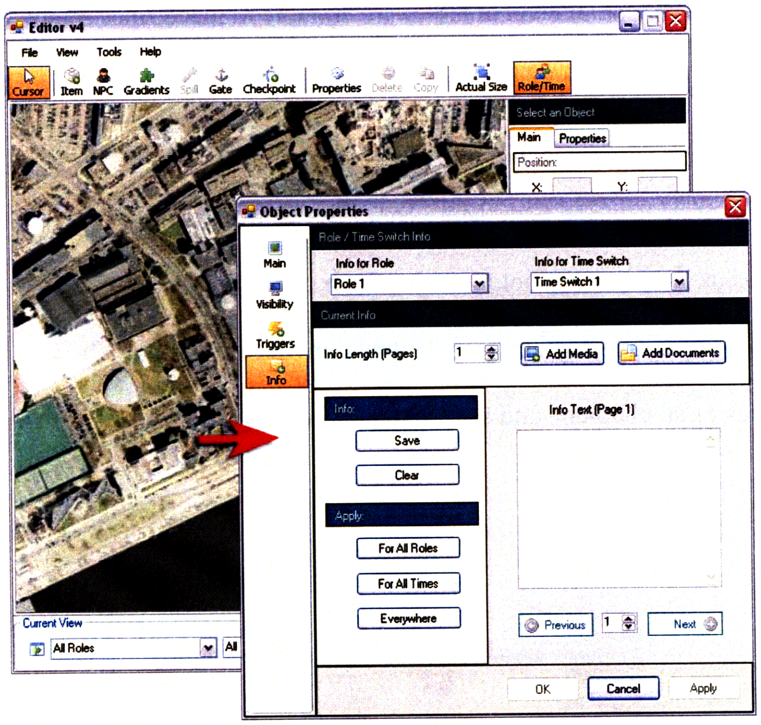


Figure 4.8 Editing Content in Game Editor. Content in the Game Editor was edited using individual property dialog forms which allowed only one object to be viewed and edited at a time.

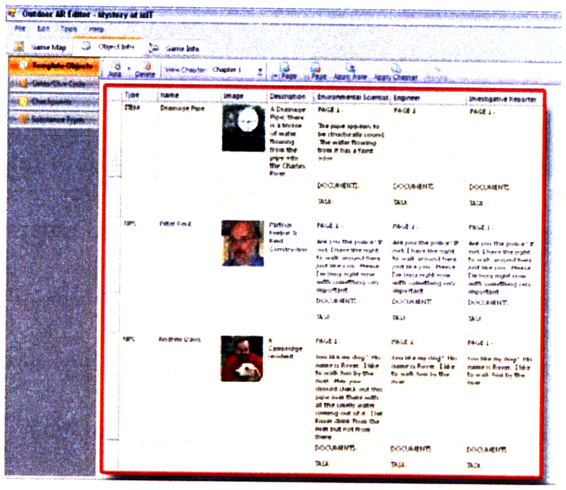


Figure 4.9 Editing Content in Desktop Editor. Content tables bring a new way of visualizing and accessing game content. The new layout should prove to be a more efficient and effective method of working with data.

Unlike Excel sheets, not all content tables are editable in place since many of the game content values information are more complex than simple strings. For example, an object dialog pages can be both plain text and media files, gate containment information are lists of selectable objects, and all game object profiles consist of images and text. More complex information fields such as these are edited in property panels associated with each content table. Each content table also has a toolbar which may include options to add or remove objects from the table, invoke an "Apply All" function to copy content into fields for all roles or all chapters, and sort content according to object type, name, or other parameters.

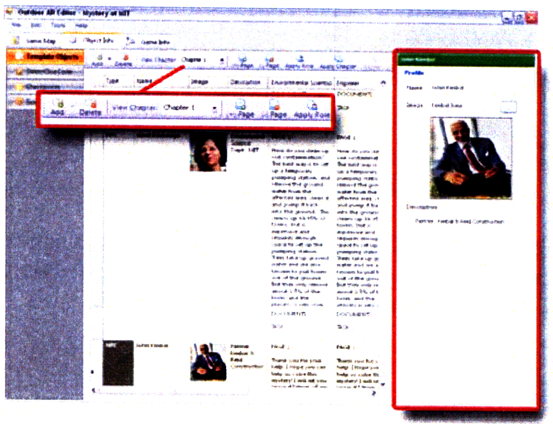


Figure 4.10 Editing Content in Desktop Editor. Each content table has an associated property panel and toolbar which is used to edit the displayed content.

The use of content tables helps a great deal in increasing the visibility of game content. Content that was previously hidden in dialog forms and could only be viewed and edited one game object at a time can now be viewed and edited simultaneously in a single location. Many game designers plan out game content using Word documents or Excel sheets. Populating game content according to these designer files becomes much more efficient since it requires fewer steps to port (copy and paste) from the designer's text editor of choice into the Desktop Editor. Imagine the process as updating a list of phone numbers that was organized in an Excel spreadsheet as opposed to a rolodex. Instead of looking up and pulling out an index card for each modification, you simply select the proper phone number cell in the spreadsheet and start editing. Spot checking content for any errors or typos also becomes a much easier process. Instead of navigating through dozens of game object dialogs and keeping careful accounts of what content has been populated and verified, designers can merely scan a single content table.

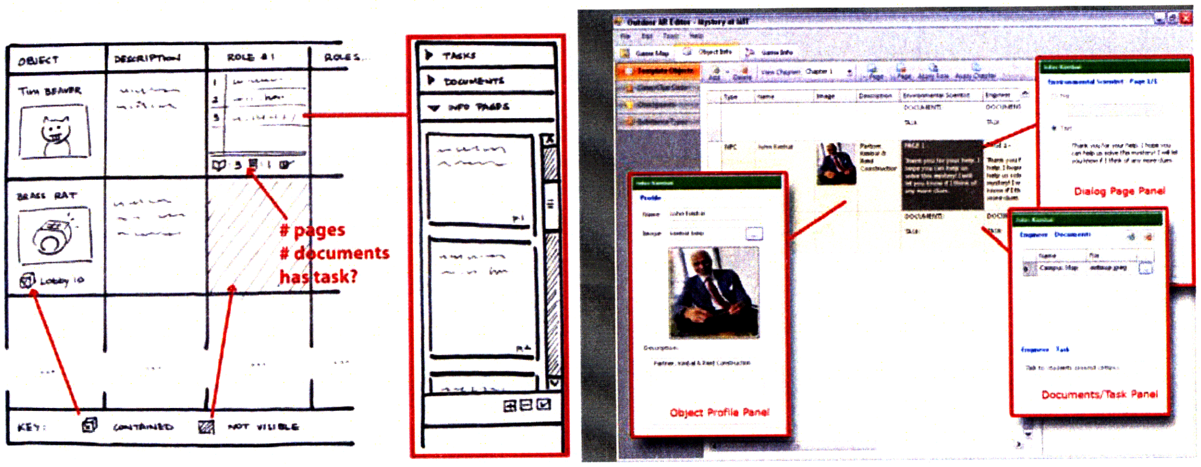


Figure 4.11 Content Table Paper Prototype and Final Design. Many of the visual complexities from original prototypes were removed from the final design to make the interface more simplistic and more in line with the modularity theme.

The final design of the content tables is not as visually robust as earlier prototypes. The original paper prototypes contained more iconic representations of information such as the visibility of an object, its containment status, and the number of associated documents. This cut back was both a limitation of implementation as well as a design decision to not overwhelm the designer with too much information.

Collapsing all game object information into a single content table would have worked contrary to the goal of modularity and the grouping of different game functionality within separate tabs and pages.

Table 4.1 Permutations of Content Table Headers.

Row Header	Column Header	Analysis
Game Object	*Role*	*- maximum visibility of information in table* *- utilizes top-to-bottom scrolling*
Game Object	Chapter	- moderate visibility of information in table (typically more roles than chapters) - utilizes top-to-bottom scrolling
Role	Game Object	- maximum visibility of information in table - utilizes side-to-side scrolling
Role	Chapter	- minimal visibility of information in table, limited to Viewing one game object at a time - utilizes side-to-side scrolling
Chapter	Game Object	- moderate visibility of information in table (typically more roles than chapters) - utilizes side-to-side scrolling
Chapter	Role	- minimal visibility of information in table, limited to Viewing one game object at a time - utilizes top-to-bottom scrolling

Another design decision was determining which dimensions the content table would be organized along. Currently game object content is specified based on three criteria: the game object, the player role, and the game chapter. The final design utilizes player roles as column headers, game objects as row headers, and chapters as selections in a drop down box which switches the content of the tables according to the selected chapter. The final design was based upon an analysis of all permutations of the dimensions from a game design and usability standpoint. Using a role and chapter combination would still have provided more content visibility than the current Game Editor but still has the limitation of viewing only one game object at a time. Since most OAR game implementations either have only a single chapter or more roles than chapters, chapters were the better candidate for the drop down box since it would be used less frequently to change views. Lastly, player roles were chosen as column headers as opposed to row headers since there are always more game objects than player roles and side-to-side scrolling is more difficult and less conventional than top-to-bottom scrolling (Accot and Zhai, 1997).

- Extended Property Panels

In addition to increasing the visibility of content, the Desktop Editor strives to increase the visibility of important game options and settings. Many of the options in the property dialog forms of the current Game Editor have been moved to more persistent property panels. The "Map" and "Game Info" tab have been fitted with collapsible property panels at the bottom of each tab which contain game map and game setup properties, respectively. These property panels contain only the most commonly used properties while more advanced settings such as Auto-zoom and Paint-a-Path map features remain in the full game options dialog form which may be accessed via the "All Properties" button on either of the collapsible panels or through the Edit menu.

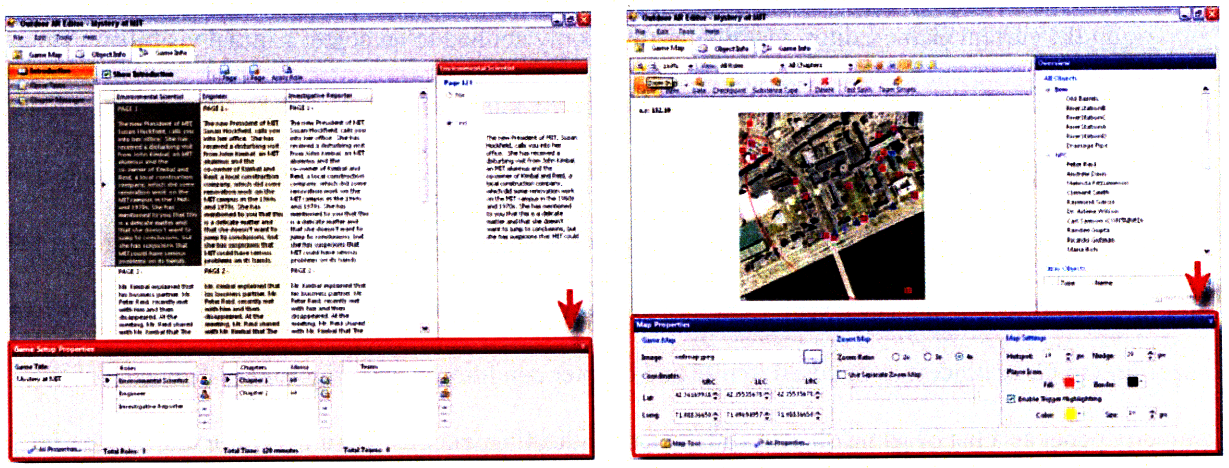
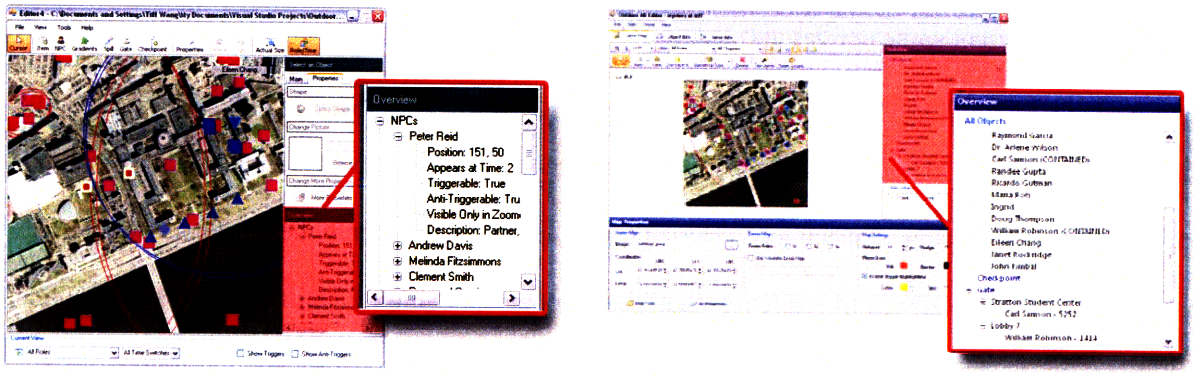


Figure 4.12 Map and Game Setup Property Panels. Persistent but collapsible property panels on the "Map" and "Game Info" tabs contain game map and game setup properties.

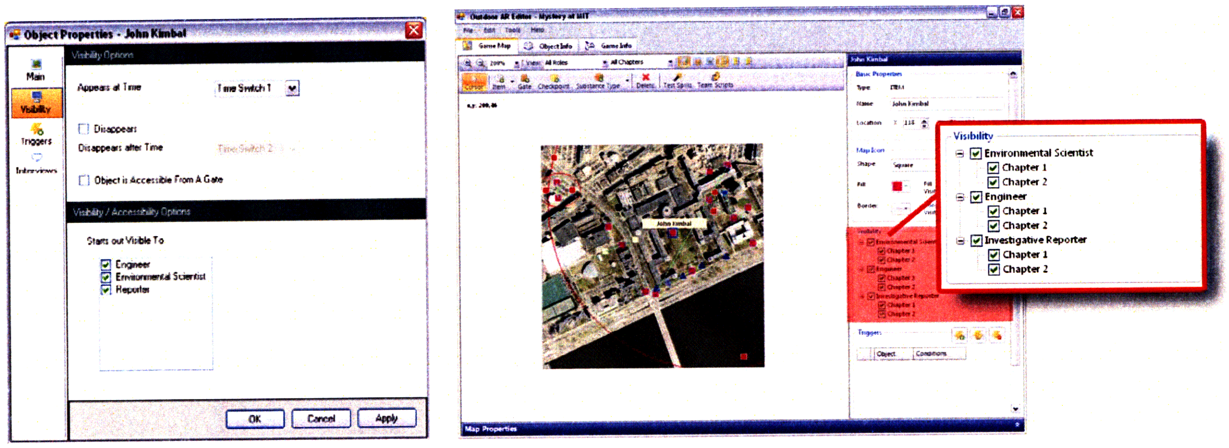
In the current Game Editor, the main interface includes a side property panel which switches between an object overview tree list and a selected game object's properties. These property panels have been kept intact but modified slightly. Extraneous information was removed from the overview tree and the tree now only displays information related to an object's location and containment status. Especially useful are the lists of contained objects displayed with each gate object since these contained objects cannot be selected from the map. For the object property panel, visibility and trigger settings are new additions to the already present basic object and icon properties.


```
Overview
⊟ NPCs
   ⊟ Peter Reid
         Position: 151, 50
         Appears at Time: 2
         Triggerable: True
         Anti-Triggerable: Tru
         Visible Only in Zoom
         Description: Partner,
   ⊞ Andrew Davis
   ⊞ Melinda Fitzsimmons
   ⊞ Clement Smith
```

```
Overview
All Objects
      Raymond Garcia
      Dr. Arlene Wilson
      Carl Samson (CONTAINED)
      Randee Gupta
      Ricardo Gutman
      Maria Kim
      Ingrid
      Doug Thompson
      William Robinson (CONTAINED)
      Eileen Chang
      Janet Rockridge
      John Kimbal
   Checkpoint
 − Gate
   − Stratton Student Center
         Carl Samson - 5252
   − Lobby 7
         William Robinson - 1414
```

Figure 4.13 Map Overview Tree List: Before and After. Extraneous information was removed from the current Game Editor's overview tree and now only includes the object's location and containment status.

In the current Game Editor, visibility settings only applied to an object's initial visibility. Thus designers could only specify when an object first appeared and disappeared. The Desktop Editor allows designers to set visibility for any chapter and for any role using a checkbox tree that lists out all player roles and chapters. Triggers were previously specified by checking off objects for a particular player role and chapter combination. The Desktop Editor takes a completely opposite approach, specifying triggers according to game objects as opposed to role and chapter conditions. The triggers are displayed in the property panel as a list of game objects with an icon indicating whether it is a trigger or anti-trigger in addition to a list of role and chapter conditions for which the trigger applies. Adding or editing a trigger brings up a dialog from which the designer may select a game object, the type of trigger, and conditions using a checkbox tree similar to the visibility settings.


Object Properties - John Kimbal
Main
Visibility
Triggers
Interviews
OK
Cancel
Apply
Outdoor AR Editor - Mystery at MIT
Map Properties

Visibility Options

Appears at Time: Time Switch 1

☐ Disappears

Disappears after Time: Time Switch 2

☐ Object is Accessible From A Gate

Visibility / Accessibility Options

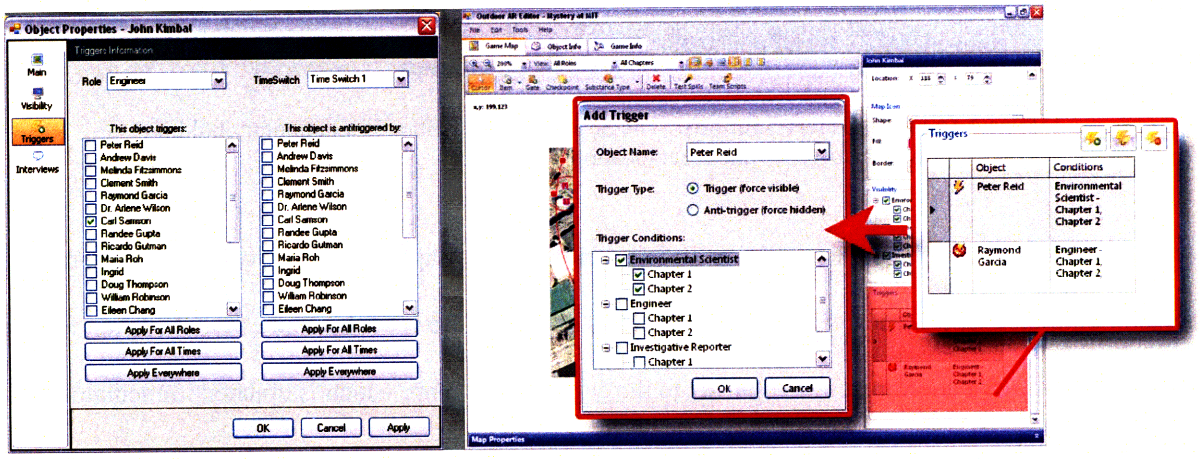
Starts out Visible To

☑ Engineer
☑ Environmental Scientist
☑ Reporter

Visibility

☑ Environmental Scientist
  ☑ Chapter 1
  ☑ Chapter 2
☑ Engineer
  ☑ Chapter 1
  ☑ Chapter 2
☑ Investigative Reporter
  ☑ Chapter 1
  ☑ Chapter 2

Figure 4.14 Editing Game Object Visibility: Before and After. The visibility properties for game objects have been moved from the object property dialog into the map property panel.


Object Properties - John Kimbal
Main
Visibility
Triggers
Interviews
OK
Cancel
Apply
Outdoor AR Editor - Mystery at MIT
Map Properties

Triggers Information

Role: Engineer   TimeSwitch: Time Switch 1

This object triggers:	This object is antitriggered by:
☐ Peter Reid	☐ Peter Reid
☐ Andrew Davis	☐ Andrew Davis
☐ Melinda Fitzammons	☐ Melinda Fitzammons
☐ Clement Smith	☐ Clement Smith
☐ Raymond Garcia	☐ Raymond Garcia
☐ Dr. Arlene Wilson	☐ Dr. Arlene Wilson
☑ Carl Samson	☐ Carl Samson
☐ Randee Gupta	☐ Randee Gupta
☐ Ricardo Gutman	☐ Ricardo Gutman
☐ Maria Roh	☐ Maria Roh
☐ Ingrid	☐ Ingrid
☐ Doug Thompson	☐ Doug Thompson
☐ William Robinson	☐ William Robinson
☐ Eileen Chang	☐ Eileen Chang
Apply For All Roles	Apply For All Roles
Apply For All Times	Apply For All Times
Apply Everywhere	Apply Everywhere

Add Trigger

Object Name: Peter Reid

Trigger Type: [✓ Trigger (force visible)] Anti-trigger (force hidden)

Trigger Conditions:

☑ Environmental Scientist
  ☑ Chapter 1
  ☑ Chapter 2
☐ Engineer
  ☐ Chapter 1
  ☐ Chapter 2
☐ Investigative Reporter
  ☐ Chapter 1

OK Cancel

Triggers

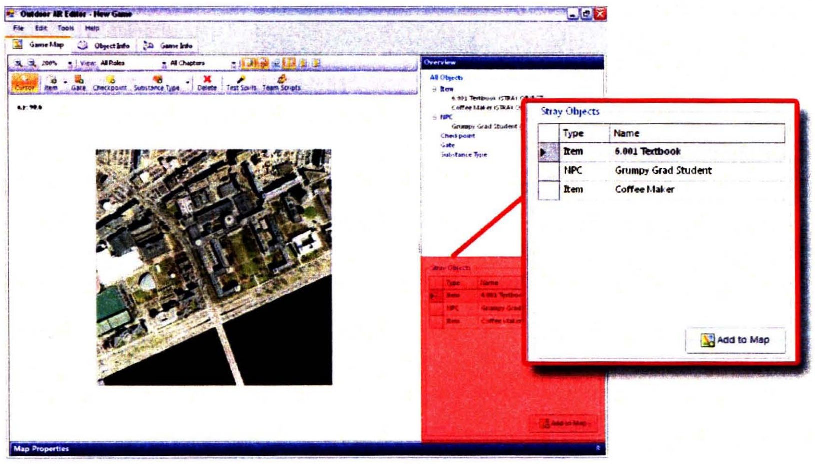
Object	Conditions
Peter Reid	Environmental Scientist - Chapter 1, Chapter 2
Raymond Garcia	Engineer - Chapter 1, Chapter 2

Figure 4.15 Editing Game Object Triggers: Before and After. The trigger properties for game objects have also been moved from the object property dialog into the map property panel.

By exposing more options that were once buried within dialog forms, the interface maximizes the visibility of important options and settings and reduces the amount of navigational traffic through the interface. Designers will be able to more efficiently find and set game options and new designers will be more aware of what options are available and important to specify for game play.

- Stray Object Sandbox

A new feature added to the Desktop Editor is the ***Stray Object Sandbox***. The sandbox is a temporary holding area for game objects with no specified location (i.e. a "stray object") and thus cannot be accurately displayed on the game map.



Type	Name
Item	6.001 Textbook
NPC	Grumpy Grad Student
Item	Coffee Maker

Figure 4.16 Stray Object Sandbox. The Stray Object Sandbox is a temporary holding area for objects without set locations.

The sandbox is displayed on the overview panel of the "Map" tab. To move an object from the sandbox onto the map, the designer simply selects a stray object from the list and clicks the "Add to Map" button. The object's coordinates are then set to the center of the map, which is updated immediately in the interface. The designer can then move and position the object at will.

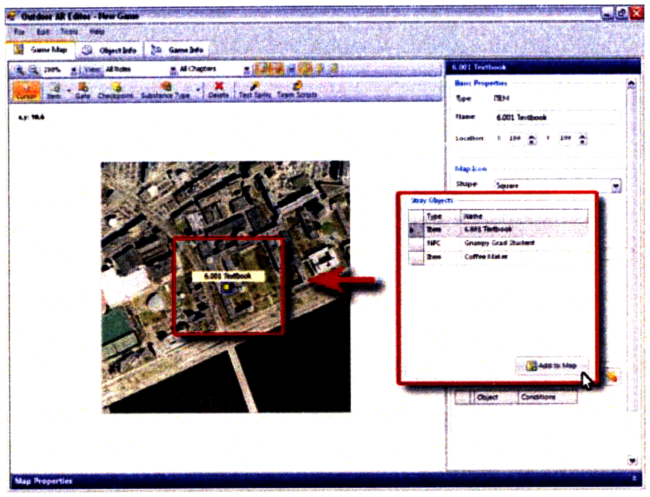


Figure 4.17 Moving a Stray Object onto the Game Map. Stray objects are placed at the center of the map by default when moved out of the sandbox.

This feature was added to the Desktop Editor since designers can now add game objects by means other than directly placing them on the game map, more specifically by adding objects to content tables on the "Object Info" tab. By placing the completed objects into a temporary holding area, designers do not have to switch back and forth between the map and the content views or worry about object placement as they populate game object content.

- Spill Testing

Another new feature added to the Desktop Editor is ***Spill Testing***. Designers have the ability to enter a "spill testing" mode on the "Map" tab to retrieve samples of spill objects using different testing methods, just as a player would when playing the game.

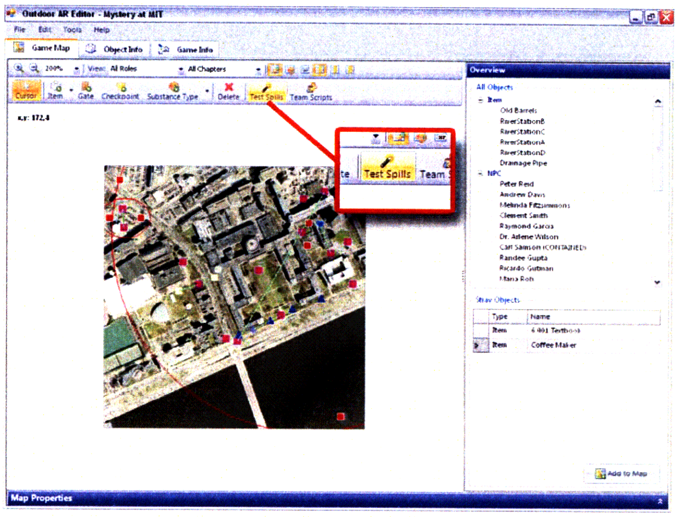


Figure 4.18 Spill Testing. The spill testing feature allows game designers to verify the placement and values of spill objects placed on the game inside the Desktop Editor.

The Spill Test buttons is located on the "Map" tab toolbar. The panel to the right of the game map allows designers to select the substance type they would like to test and the test method they would like to use for sampling. Samples can then be retrieved by clicking on different locations of the game map. An icon labeled with the substance type and sample value will be displayed on the map.

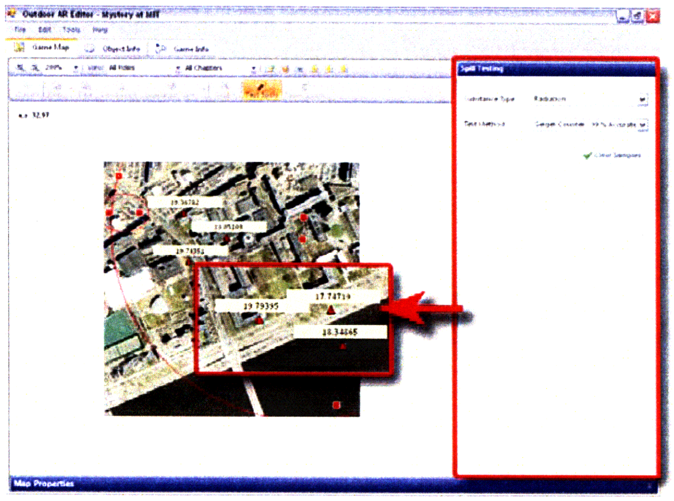


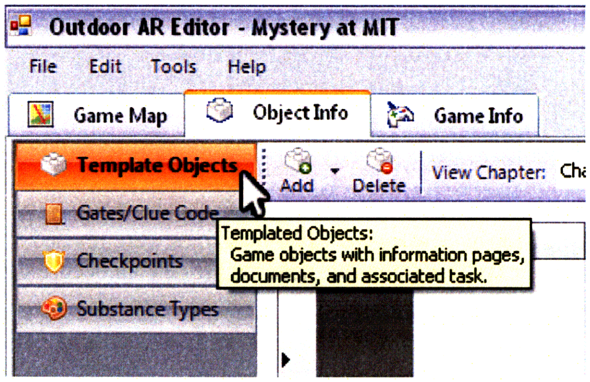
Figure 4.19 Testing Spill Object Values. Clicking on various locations of the map will display icons labeled with the substance type and sample value.

This feature was added to help designers test placement and sample values of spill objects on the game map. Though it may be more accurate in terms of game play to perform sample testing on location, designers may not always have the time to walk around the entire game map taking samples and plotting values for verification. This feature provides a quick and easy way to test samples and catch common mistakes such as incorrect maximum intensity values, incorrect drop off rates, or overlapping spill objects which return abnormally high values since the values are additively compounded.

- More User Feedback

As part of the cleanup effort of the Game Editor, a few passes were made through the existing interface focusing on providing designers with more information about editor features and functionality.

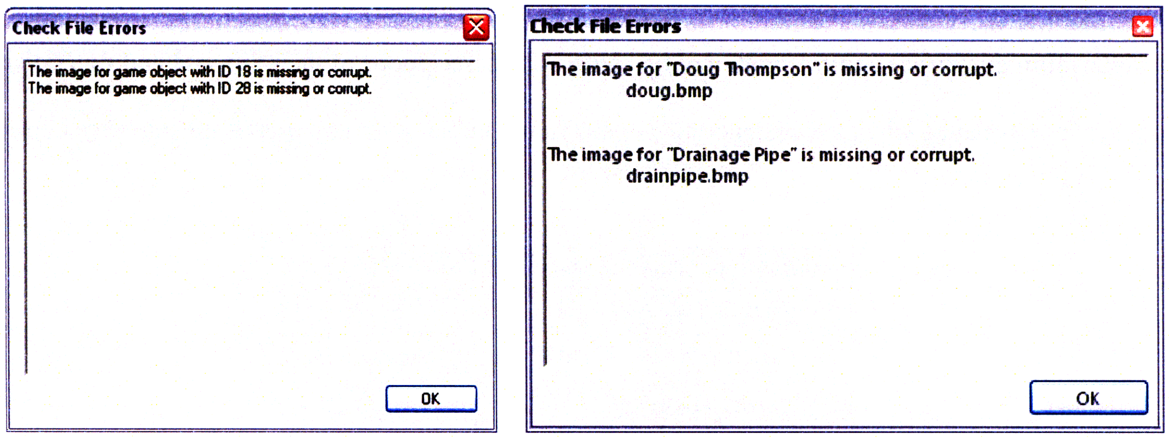
The first pass focused on clearer and more consistent naming and labeling standards for features as well as fitting interface controls with tool tips with brief descriptions of their purpose and functionality. Tool tips are displayed when the designer hovers the mouse cursor over any part of the interface control.



Templated Objects:
Game objects with information pages, documents, and associated task.

Figure 4.20 Interface Tool Tips. The majority of interface controls are fitted with tool tips explaining the purpose of each feature and how it should be used.

A second pass through the interface focused on clarifying error messages and placing more confirmation dialogs in riskier parts of the interface. A number of the error messages in the current Game Editor contain cryptic messages and jargon which do not divulge much information about the cause of the error and how to resolve it. One example is the File Checker which checks for missing or broken game content file paths and displays a list of problematic links. The current Game Editor returns a list of IDs for media files that have broken paths which is not very helpful for resolving the issues since media IDs are not provided to the designer anywhere else in the interface. This particular example was redesigned to display the name of the game object that the missing file is associated with as well as the full path of the broken link. Confirmation dialogs were also placed in many locations that were at risk of designers accidentally deleting a lot of information, such as removing roles or chapters which would delete all associated game content. Although pop up dialogs can be a bit of an annoyance, it is less tragic than recovering hours worth of lost work.



```
The image for game object with ID 18 is missing or corrupt.
The image for game object with ID 28 is missing or corrupt.
```

```
The image for "Doug Thompson" is missing or corrupt.
        doug.bmp

The image for "Drainage Pipe" is missing or corrupt.
        drainpipe.bmp
```

Figure 4.21 Helpful Error Messages. Error messages were revised to help explain the problem and propose possible solutions for errors occurred during game editing.

Another pass focused on identifying property fields which could use default values when being newly added. Examples include designating new roles as "New Role" or defaulting pages added to an object's dialog to have the text "New Page". Default values provide new designers with additional clues on how to use features by providing examples of what information should be placed in a particular field.

Default values are also good indicators of missing data. A common mistake that designers make is confusing missing data with broken functionality. For example, if a designer forgets to specify dialog text for a particular role or chapter, no dialog is displayed for that game object in game since the OAR Game Engine is defaulted to ignore blank pages. Game designers must now intentionally leave a page blank, otherwise the text "New Page" will appear in game.

4.3 Implementation

The Desktop Editor is a Windows application implemented in C# using the Microsoft .NET Framework SDK 2.0. An additional development SDK, the Krypton Component Factory SDK 2.8.5, was also used in order to have access to an extended library of user interface controls. The structure of the code base for the Desktop Editor is organized by modules according to the Model-View-Controller development model (Apple Inc., 2007).

- Model

The Model module of the Desktop Editor is a collection of OAR game-related object classes. Much of this code was previously maintained in separate code bases for the Game Engine and Game Editor. In order to reduce the amount of redundant code as well as the sections of code that needed to be updated whenever changes were made to the game, the object classes were recently refactored so that all classes reside in a single library, the ***CommonLib***, which is shared between all the OAR applications.

One notable change to the backend code during the refactoring is the new structure of game object types. Items, NPCs, Gates, and Checkpoints were previously specified as sub-classes of a base object class but largely handled their properties and functionality in their own derived classes. The game objects in CommonLib are all instances of the same ArTemplateObject class and utilize a templating system to specify unique values for common game object properties as well as to specify special behaviors such as providing dialog information (i.e. Items and NPCs) or containing other game objects (i.e. Gates). The main motivation behind the use of templates was to provide a way of visually customizing game objects that are functionally the same. For example, Items and NPCs are essentially the same game object in code but are referenced by different names and may have different map icons for the purposes of game play

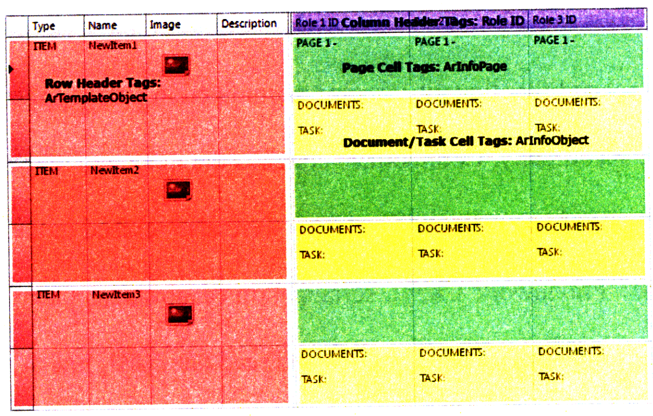
Table 4.2 Game Object Base and Template Properties.

Game Object Base Properties	
ID	unique object ID number
Visibility Conditions	conditions object is visible
Trigger Events	objects triggered by this object & conditions
AntiTrigger Events	objects anti-triggered by this object & conditions
Location	x,y coordinates of game object
Image Path	profile image path
Template Object Properties	
Glyph Shape	map icon shape
Glyph Fill Color	map icon fill color
Glyph Border Color	map icon border color
Glyph Visited Fill Color	map icon fill color (visited)
Glyph Visited Border Color	map icon border color (visited)
Template Icon Image	template icon image for Editor UI
Show Info Panel	does game object need info property panel?
Show Access Code	does game object need code property panel?
Show Container Panel	does game object need container property panel?
Template Name	name of template type

- View

The View module of the Desktop Editor consists of all the window and dialog forms that make up the Desktop Editor interface. Since editor options are grouped according to similar game functionality, the interface forms naturally are organized in a similar structure: layered pages and tabs. Each of the forms is responsible for handling user input, displaying the proper game data, and ensuring that the data is in sync with the backend.

The most taxing part in implementing the interface was populating and maintaining the data displayed in content tables. In the current Game Editor, game content is accessed in dialog forms on an as needed basis, which simplifies the display and editing of data. Content tables display all game content simultaneously, allowing the data to be viewed and edited at will. The interface controls tagging system provided by the .NET Framework was utilized to simplify data maintenance. Every row header, column header, and cell was tagged with object IDs or pointer references to data so the values of that data could be accessed and updated without having to perform excessive look ups. As a result a lot of accounting had to be done to make sure tags were correct and that data was being correctly referenced and modified.



	Type	Name	Image	Description	Role 1 ID	Column Header Tags: Role ID	Role 3 ID
	ITEM	NewItem1			PAGE 1 -	PAGE 1 -	PAGE 1 -
					DOCUMENTS: TASK:	DOCUMENTS: TASK:	DOCUMENTS: TASK:
	ITEM	NewItem2					
					DOCUMENTS: TASK:	DOCUMENTS: TASK:	DOCUMENTS: TASK:
	ITEM	NewItem3					
					DOCUMENTS: TASK:	DOCUMENTS: TASK:	DOCUMENTS: TASK:

Row Header Tags: ArTemplateObject
Page Cell Tags: ArInfoPage
Document/Task Cell Tags: ArInfoObject

Figure 4.22 Content Table Tagging. In order to provide faster access to game data and avoid excessive lookups, row headers, column headers, and table cells were tagged with different game data references. This figure shows the tags used on one of the more complex tables: Item and NPC information.

Furthermore, content table cells contain varied types of data which are edited in different manners. The proper property panel had to be created and displayed based on the cell selected in the content table in order to provide the designer with access to the right editable fields. In retrospect utilizing the SDK's Property Grid control would have provided a simpler and more uniform way of modifying property values.

- Controller

The Controller module acts as an intermediary between the toolkit user interface and the backend data. Since much of the backend data can be accessed directly with simple set methods and the View classes directly handle a good portion of processing user input, the Controller module of the Desktop Editor is fairly small. The EditorManager class is the main Controller class and primarily handles more computationally expensive methods for accessing and manipulating data that are referenced in several locations in the Desktop Editor. Some of the tasks that the EditorManager takes care of include performing clean up work when game objects and other settings are removed from the game, and tracking changes made to game data to ensure different parts of the interface are updated appropriately to stay in sync.

Some Controller related functionality, is also maintained in CommonLib since it is used by all OAR applications. More specifically, CommonLib contains the input and output utility classes used for saving and loading OAR game files. The utility classes were also revised in the recent code refactoring to take advantage of the .NET Framework's XML serialization capabilities. Game files are now loaded and saved using serialization as opposed to being written and read one XML line at a time. This change also helps reduce the amount of code that needs to be maintained when changes are made to the game.

5. Remote Editor

The Remote Editor is the handheld side of the new OAR Game Editor toolkit. A brand new tool being introduced to OAR game creation, the Remote Editor was designed as a mobile compliment to the Desktop Editor. The intended use of the application is to allow designers to directly modify and test game implementations while at the game's physical location.

This chapter documents the design and implementation process of creating the Remote Editor. Development goals, prototype screenshots, and implementation details are presented in the following sections.

5.1 Development Goals

The development of the Remote Editor was largely motivated by the importance of game locality as part of the concept of OAR games. In order for an OAR game to be an effective and truly immersive experience, the game must be conscious of the characteristics of a location and try to make connections between objects in the location and game content as much as possible. By creating a mobile editing tool, the game location can more easily be incorporated into the OAR game design process. Game designers are more likely to spend time exploring the location while planning their games and to more accurately work objects in the game location into the content.

Another motivation for the Remote Editor was expediting the process of testing game play. All problems and information gathered from pilot testing a game must be carefully documented, manually updated in the Game Editor, and verified again at a second trip to the game site. This multi-step process requires of lot of time and effort, especially if the game location is not easily accessible.

Following from these motivations, there are two major tasks that the Remote Editor application focuses on: (1) location-based game design and (2) on-site game refinement. Location-based game design allows designers to immediately add, remove, or modify game objects and log memo notes as they are exploring the location. On-site game refinement allows designers to test game object placement and game settings and directly make adjustments as necessary.

5.2 Prototypes & Design

This section contains prototype sketches and design descriptions for the Remote Editor, organized according to feature. The interface design and general functionality of the Remote Editor is heavily based off the Game Engine application. The reason for mirroring the Game Engine is to provide experienced game designers with a familiar interface and to keep a consistent look and feel across the OAR game applications.

- Remote Editor Setup

To start editing a game in the Remote Editor, the designer loads a game file into the application the same way a game file is loaded into the Game Engine: using a file selection screen at launch time. The game file can contain as much or as little game content as the designer wishes to work with but must have the bare minimum of one role, one chapter, and a game map with valid map coordinates.

Two methods were considered for setting up games in the Remote Editor. The first method was starting a game completely from scratch. The designer would have to acquire a map using the Desktop Editor's Map Tool, load the image and corresponding coordinate file on to the handheld device, reference those map files when starting up the Remote Editor, and finally save it into a playable game file. The second and current method was building on an existing OAR game file and involved the steps described at the beginning of this section.

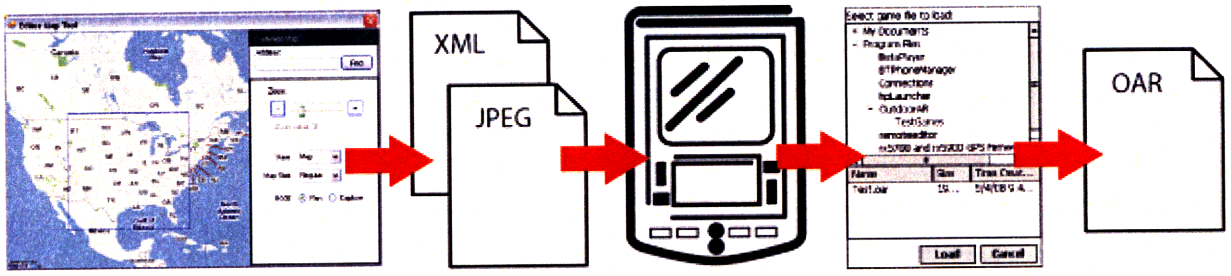


Figure 5.1 Scratch Method Setup. Using the scratch method for setting up the Remote Editor requires acquiring a map using the Map Tool, uploading the image and coordinate file to the handheld, loading the files in the Remote Editor, and saving the game into a playable game file before beginning to edit.

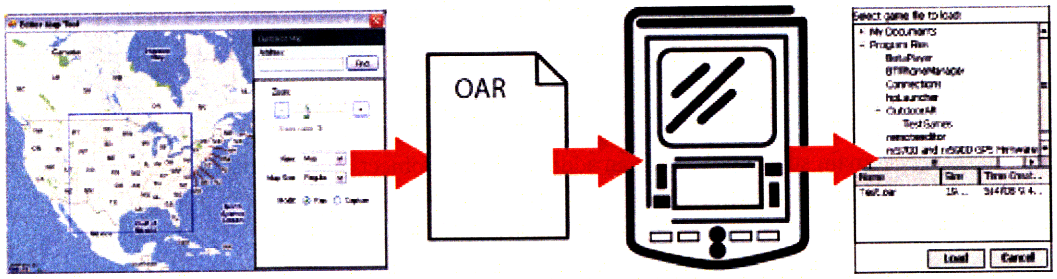


Figure 5.2 Build Method Setup. Using the build method for setting up the Remote Editor requires initializing a game file in the Desktop Editor, uploading the file to the handheld, and loading the files in the Remote Editor to start editing.

The build method is the simpler of the two methods in terms of number of steps required to set up the Remote Editor as well as consistency with loading games in the Game Engine. Since the Remote Editor also allows designers to edit and test pre-existing game files, it needs to support the loading of OAR game files regardless. Maintaining a single loading and setup sequence for both new and existing games is more consistent and less confusing for users.

- Main Interface

Given that the Remote Editor puts a heavy emphasis on game location, the main interface of the Remote Editor is naturally dominated by the game map. The main screen consists of several parts: map panel, GPS status bar, editing toolbar, and the file menu bar. The original prototype design also contained a game object information panel which displayed the profile information of a selected game object, but due to limitation on screen space, this feature was redesigned as a caption box displayed on top of the game map.

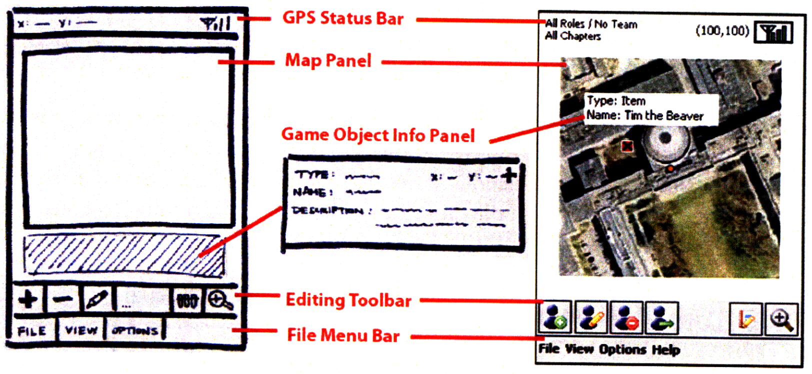


Figure 5.3 Main Screen Original and Final Prototypes. The original mockup of the main screen contained an additional game object information panel which was changed to a caption in order to save screen real estate.

Like the Game Engine main screen, the map panel displays the game map with icons indicating the placement of all game objects as well as an icon tied to the location of the designer as he moves around the map.

The GPS status bar is an up-to-date display of the game designer's location in game coordinates and the signal strength of the GPS device. Since the designer does not normally need to be concerned with the finer details of the GPS interface such as satellite count or degree of precision, the status bar displays just the essential amount of GPS information. More detailed GPS information may be found on the GPS Status screen which is accessible via the Options menu.

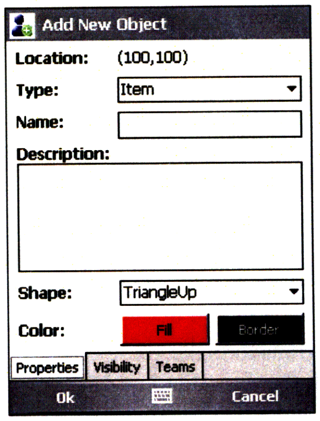
The editing toolbar provides direct access to the main features of the Remote Editor: adding, editing, deleting, and moving game objects, taking notes, and zooming in and out of the game map.

These features were placed on the main screen since they are related to map activity and the designer must be able to view the game map while using these features. The toolbar uses iconic buttons in order to save screen space, but mainly to provide more appealing visual cues for each button's functionality.

- Game Editing

The Remote Editor allows game designers to edit a subset of all game options and settings. Designers may add and delete game objects, change the locations of game objects on the map, modify basic properties of each game object, and tune map-related game settings.

Adding a new object to the map or selecting an existing object to edit brings up the Object Properties screen. The Object Properties screen covers only the most basic game object properties such as type, location, icon look-and-feel, name, description, and visibility conditions for roles, chapters, and teams. This subset of properties is sufficient for placing an object on the map, specifying a name and map icon to identify the object, and providing a short descriptive note for the game object. All text content more complicated than a short description such as dialog text was not intended to be edited with the Remote Editor since the tap-to-type keyboard input method on handheld devices is not well suited for performing large amounts of text editing.


Add New Object

Location: (100,100)

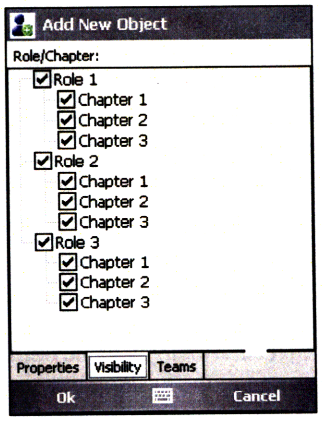
Type: Item

Name:

Description:

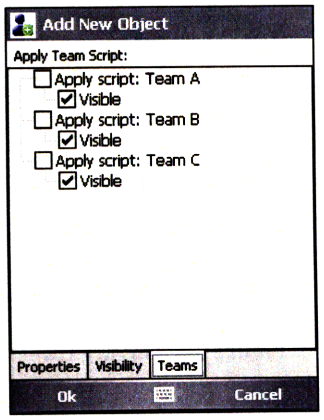
Shape: TriangleUp

Color: Fill Border


Add New Object

Role/Chapter:

- ☑ Role 1
  - ☑ Chapter 1
  - ☑ Chapter 2
  - ☑ Chapter 3
- ☑ Role 2
  - ☑ Chapter 1
  - ☑ Chapter 2
  - ☑ Chapter 3
- ☑ Role 3
  - ☑ Chapter 1
  - ☑ Chapter 2
  - ☑ Chapter 3


Add New Object

Apply Team Script:

- ☐ Apply script: Team A
  - ☑ Visible
- ☐ Apply script: Team B
  - ☑ Visible
- ☐ Apply script: Team C
  - ☑ Visible

Figure 5.4 Object Properties Screen. The Object Properties screen allows the game designer to set a game object's type, name, description, icon look-and-feel, and visibility according to role, chapter, and team.

Deleting or moving objects is done directly from the main screen by selecting a game object and tapping on the delete or move button, respectively. When a designer moves a selected game object, the object is moved from its current location to the designer's location on the maps. Moving game objects was one of the more open-ended design questions while implementing the Remote Editor. One moving mechanic considered was using the handheld device's thumb pad controls to move the object up, down, left, or right. However, this was a slower and more inefficient method since the game object would have to translated and redrawn on the map for every small pixel increment. A second method involved first selecting the game object, then tapping on the move button, and finally tapping on a new map location to move the object. This "tap-to-move" method was originally implemented and later disabled and replaced by the current "move-to-me" method. Based on initial evaluations by the project development team, it was determined that the tapping method was a less fluid mechanic and that the "move-to-me" method would be more useful since in most cases the designer would already be at or near the desired location of the game object.

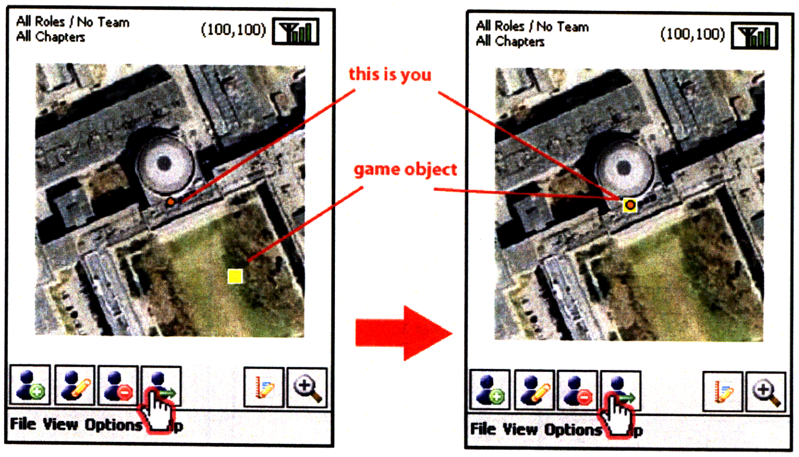
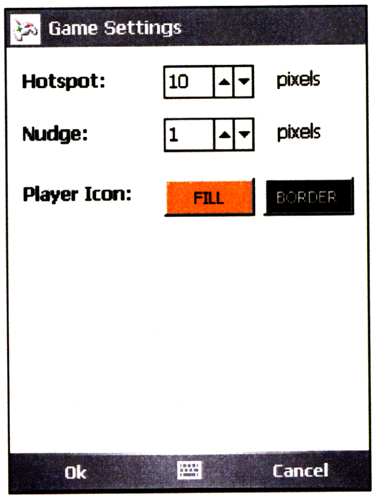


Figure 5.5 Moving Game Objects. Moving a game object transports the game object from its current location to the location that the designer is standing at.

Game settings are viewed and edited on the Game Settings screen which may be accessed from the Edit menu. The current editable settings include only map settings such as the player icon color and game object visitation settings such as hotspot and nudge size. Hotspot size is the range around a game object in which a player can bump and "visit" the object. Similarly, nudge size is the range around a game object in which a player can tap to "visit" the object.


Game Settings

Hotspot: 10 pixels

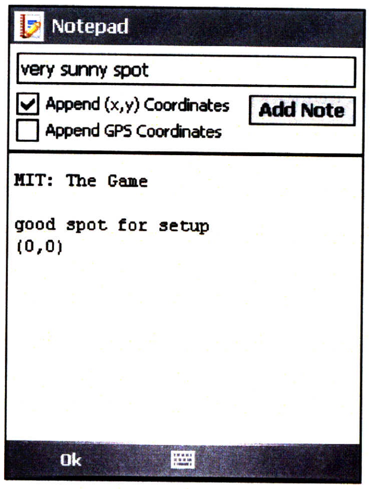
Nudge: 1 pixels

Player Icon:

Figure 5.6 Game Settings Screen. The Game Settings screen allows the designer to modify map and game object related settings.

- Notepad

Designers will often take copious notes while exploring a game location or testing a game implementation on-site. The Notepad feature is essentially a digital scratch pad that designers can use to take notes instead of the traditional pen and paper method. In addition to text, designers have the option of automatically appending game or map coordinates to a note if the note is tied to a specific location. For example, a designer might note that a picnic table located at coordinates (25, 50) had exceptionally good GPS signal strength and would be a good place to set up the handheld devices and deliver game instructions.



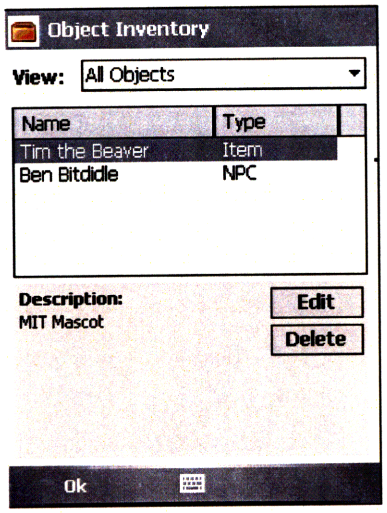
```
MIT: The Game

good spot for setup
(0,0)
```

Figure 5.7 Notepad Screen. The Notepad screen displays all entries that have been added to the note log and allows the designer to append new notes with map and game coordinates.

The Notepad is a running text log which may be viewed in the Remote Editor by tapping on the Notepad button on the mainscreen. The log is stored as a text file which can be extracted from the game file directory. Although designers can also take notes by tagging locations with pseudo game objects, the Notepad offers a faster, more collective method of gathering information from the game site.

- Inventory

The Inventory feature provides a compiled listing of all game objects currently in the game. The list of game objects is filterable by object type and may be sorted alphabetically by column. Selecting an object will populate the information panel below the table. This information was not included as additional columns in the table because the narrow width of the screen would have required lots of side scrolling between columns in order to view different property values.


Object Inventory

View: All Objects

Name	Type
Tim the Beaver	Item
Ben Bitdidle	NPC

Description:
MIT Mascot

Figure 5.8 Inventory Screen. The Inventory screen displays a list of all game objects currently in the game which can be filtered and sorted by type and name.

Since not all game objects can be simultaneously selected on the mainscreen map panel, it can be difficult to locate a particular game object if there are a large number of game objects displayed on the map. The Inventory provides the designer with a holistic and more organized view of all the game objects. The Inventory screen also provides access to the Object Properties screen so game objects may be edited from within the Inventory.

- Testing Games

As mentioned in the task analysis in Chapter 3, the task of on-site game testing involves verifying and adjusting the position of game objects and game play settings such as the size of hotspot and nudge visitation zones. Similar to playing the game with the Game Engine, designers can walk around the game map and "visit" objects by walking into their hotspots. However, instead of displaying the object's full profile, an information caption with the object's information is displayed on the map. The caption uses a darker background to indicate that the object was selected by means of "visiting" and not tapping.

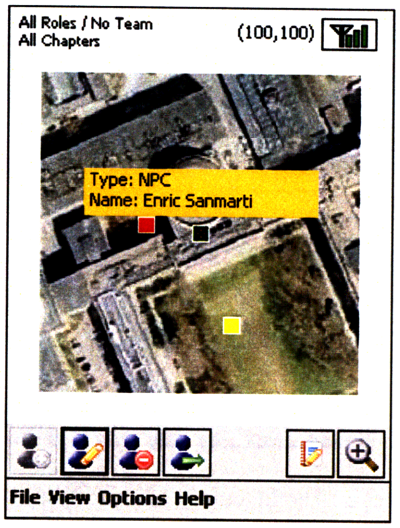


Figure 5.9 Testing Object Visitation. Walking into the hotspot of a game object will trigger a "visit" like during game play but brings up the game object info caption. The darker background color of the caption indicates that the object was selected via "visiting".

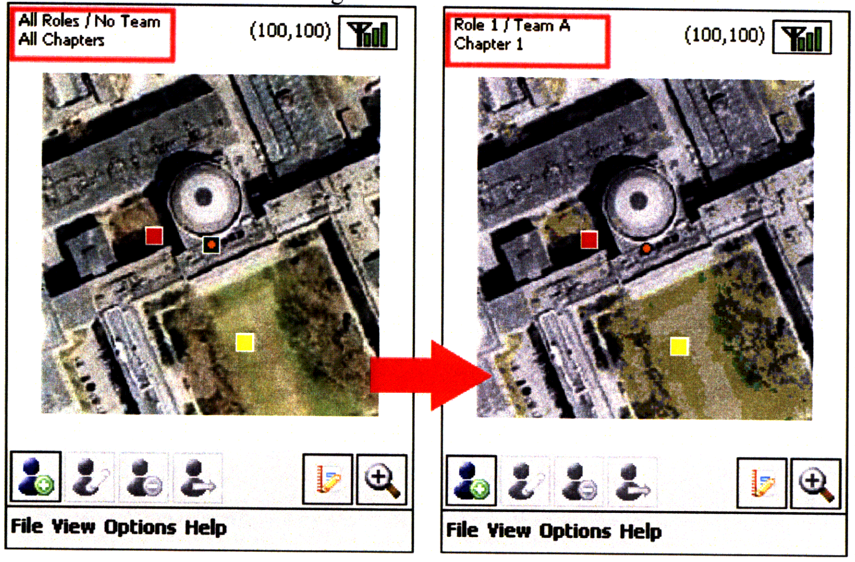
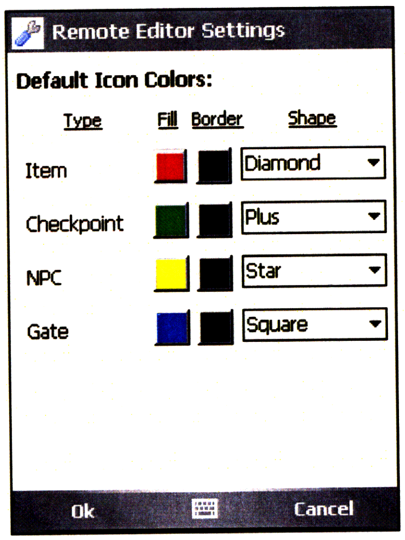


Figure 5.10 Filtering Viewable Objects. Similar to the Desktop Editor map interface, the visibility of game objects on the map can be filtered according to role, chapter, and team by using the options in the View menu.

This ability to "visit" game objects in the Remote Editor can be disabled in the Options menu if a designer does not wish to bump into objects as they are walking around the game location. Designers also have the option of filtering the display of game objects according to role, chapter, and team using the options under the View menu. This allows designers to test groups of game objects according to specific role, chapters, and team script conditions.

- Editor Settings

In addition to game settings, the Remote Editor also has editable settings for the application itself. The editor settings currently only include adjustable default values for game object type icon look-and-feel, but may be extended if more editor settings are later required.



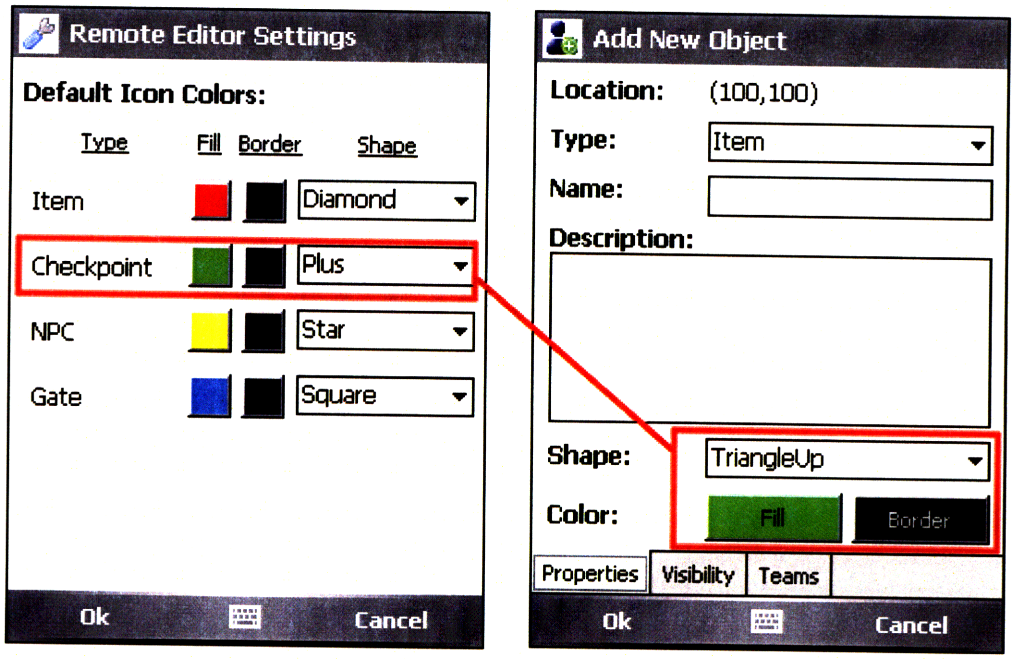
Remote Editor Settings

Default Icon Colors:

Type	Fill	Border	Shape
Item	🟥	⬛	Diamond
Checkpoint	🟩	⬛	Plus
NPC	🟨	⬛	Star
Gate	🟦	⬛	Square

Figure 5.11 Editor Settings Screen. Default icon shape and colors for game object types can be set up in the Editor Settings screen.

Adjustable default values for game object icons are a convenience feature which was created during prototype testing of the Remote Editor. Icons of game objects can be set to different colors and shapes so they can be distinguished more easily on the game map. For example Items can be displayed as red stars and NPCs as green triangles, as opposed to all game objects being displayed as a sea of blue squares. The shape and colors of game object icons on the Object Properties screen were previously initialized to the values specified by the object template files. Theses initial values can now be set and saved under the Editor Settings screen. Thus when designers are adding game objects, they do not have to perform the extra steps of remembering which shape and color they had matched with each object type and extra steps of setting the properties since the values are automatically populated when a specific type is selected.



Default Icon Colors:

Type	Fill	Border	Shape
Item	🟥	⬛	Diamond
Checkpoint	🟩	⬛	Plus
NPC	🟨	⬛	Star
Gate	🟦	⬛	Square

Location:	(100,100)
Type:	Item
Name:	
Description:	
Shape:	TriangleUp
Color:	Fill Border

Figure 5.12 Default Icon Settings. Default icon settings are saved on the handheld device so do not need to be set each time the Remote Editor is launched. Changes are immediate reflected in the Object Properties screen.

5.3 Implementation

The Remote Editor is a Windows mobile application implemented in C# using the Microsoft .NET Compact Framework SDK 2.0. The application is compatible with handheld devices that are fitted with Windows Mobile 5.0 or later and have GPS tracking capabilities (either built-in or connected through peripherals). For this particular project, the Remote Editor was installed and tested on Dell Axim x51, HP iPAQ rx5915 Travel Companion, and Pharos 525 and 535 handheld models.

Like the Desktop Editor, the structure of the code base for the Remote Editor is organized by modules according to the Model-View-Controller development model. The Model module is the same CommonLib shared among all OAR application. The View module again consists of all screens that make up the Remote Editor interface. The Controller module of the Remote Editor mainly handles monitoring and processing of received GPS data.

The Remote Editor is essentially a hybrid of the Game Engine and Game Editor, combining GPS tracking and real time player-game object interactions with the ability to modify backend game data. Although the Remote Editor and Game Engine applications are deployed using the same method and share common user interface elements, the Remote Editor was developed as a separate application to keep the interface of both applications simple and maintain separate focuses on game-editing and game-play. Combining the two applications would have required additional setup screens or additional modes in the Game Engine, adding more complexity to the application's interface.

The Game Engine is used primarily by young students who may unintentionally (or intentionally at times) stray into parts of the interface where they should not be accessing. For example, exiting the OAR application and playing games installed on the handheld. A feature to hide all non-game play related options in an "Admin Mode" screen was recently requested and implemented in the Game Engine. Implementing the Remote Editor as a separate application from the Game Engine avoids the risk of curious students modifying game data without requiring game designers to jump through extra hoops like punching in admin codes to make modifications to their games.

One of the biggest challenges when developing the Remote Editor was working around the disparity between developing a Windows desktop application and a handheld application. The .NET Compact Framework is similar to the .NET Framework used to develop Windows applications but has a much more limited toolset. A lot of features and functions available for Windows application development is not supported on the mobile. Thus a lot of missing functionality such buttons with images and graphics transformations required creating custom interface controls or even deferring features. One feature that did not make it into this iteration of the Remote Editor is viewing and testing spill objects due to the lack of support for rendering rotated images.

Another challenge faced when developing the Remote Editor was the limitation of screen real estate. Many design prototypes such as the layout of the main screen had to be scaled back and redesigned in order to find other means of displaying information that did not fit on screen. In the end, this limitation may have been beneficial to the overall design because it prevented the design from getting too overly complex and forced additional evaluations to identify the most necessary features and functionalities to be included in the interface.

6. Evaluation

This chapter documents the evaluation process of the new Desktop Editor and Remote Editor applications. The first section is an overview of the usability limitations previously identified in the existing Game Editor and an assessment of how each of the editors addressed these limitations. The second section documents the various methods used to evaluate the applications throughout the development process and the last section presents the results and findings gathered from those evaluations.

6.1 Requirements Assessment

As mentioned in the Problem Statement, this project focused on improving the usability of the existing Game Editor according to a standard set of usability measurements: learnability, memorability, efficiency, errors, and satisfaction. The following tables are organized by measurements and provide an overview of the proposed solutions for the limitations identified in the existing Game Editor. "+"s indicate methods used by the new editor applications to address each limitation and to improve general usability for each measurement, "x"s indicate areas for improvement. "DE" refers to features in the Desktop Editor, while "RE" refers to features in the Remote Editor. Since satisfaction is closely tied to the other usability criteria, it is not included in the assessment.

Table 6.1 Learnability and Memorability Assessment Overview.

Learning and Memorability	
Lack of documentation	
+ created application documentation (RE, DE)	documentation for both editor applications have been created and kept up to date
+ new game wizard (DE)	descriptions provided for basic game options needed for setup of a new game
+ tool tips (DE)	descriptions attached to user interface controls throughout the application, amount of detail provided varies with control and feature
x in-application help menu (DE, RE)	
Insufficient interface cues	
+ tool tips (DE)	hints about the purpose and at times instructions for the control displayed when mouse cursor hovers over interface controls
+ iconic buttons (DE, RE)	icons on buttons used as visual cues for the purpose of certain features
+ revised feature labeling (DE, RE)	more consistent naming of features used in applications, more ambiguous features renamed to provide better insight on purpose and functionality
+ new interface organization (DE)	editor options grouped by related functionality to provide cues on how features are used and how they fit into the scope of a game
+ mapping between content table cells and property panels (DE)	content table cells are associated with a particular property panel which includes only options related to the contents of the selected table cell
Inconsistent interface	
+ start up screen (DE)	start screen displayed at launch similar to many other editing applications
+ new interface organization (DE)	layout of interface controls particularly the tabs, pages, and content tables are standardized across the application interface
+ revised feature labeling (DE, RE)	more consistent naming of features to avoid any ambiguities when being referenced
Other	
+ property panels (DE)	more fully populated and more persistent property panels increase visibility and access of important options
+ default values (DE)	default values provide examples of valid user input

Table 6.2 Efficiency Assessment Overview.

Efficiency	
Restrictive modal-dialog interface	
+ content tables (DE)	content tables provide holistic method of viewing and editing game content
+ new interface organization (DE)	editor options are grouped into tabbed pages instead of separate dialog forms and provide direct access to many of the game options
Lack of functionality shortcuts	
+ file menu shortcuts (DE)	keyboard shortcuts connected to many file menu options (i.e. starting new games, saving games, etc.)
+ content table context menus (DE)	content tables fitted with context menus listing options related to that particular table
x addition revision and additions needed for shortcuts and context menus (DE)	
Lack of default user input	
+ default field values (DE)	default values are set when instances of new objects or options are added to game (i.e. player roles, dialog pages, etc.)
+ new game template (DE)	starting new games loads in default game template with the bare minimum of game options already set to default values
x provide pre-made, fully populated templates for game objects and games (i.e. load an "Adventure Game") (DE)	
Non-mobile application	
+ Remote Editor implementation (RE)	Remote Editor provides ability to edit and test game implementations while on-location
Other	
+ property panels (DE)	more fully populated and more persistent property panels provide faster access to important options
+ editor settings (RE)	game object icon look-and-feel can be populated according to user specified default values

Table 6.3 Errors Assessment Overview.

Errors	
Forgetting game content	
+ content tables (DE)	content tables provide an overview of all game content and a faster, more reliable way of checking for missing game content
Forgetting game flow settings	
x timeline view of game object connections (i.e. triggers, anti-triggers) (DE)	
x logic checker: verify missing links (i.e. object triggered by invisible object) (DE)	
Formatting text and images incorrectly	
+ property panel text boxes (DE)	text boxes in game object property panels sized to match text display on handheld devices
+ content table profile images (DE)	content tables display profile images of game objects, although not in the same size as on handheld devices
x preview of text and content in handheld device view (i.e. emulator) (DE)	
Other	
+ error messages and verification dialogs (DE)	error messages fitted with clearer messages and remediation instructions, verification dialogs placed in areas of interface at risk of deleting lots of data
+ default values (RE)	default values for newly added game data provides noticeable errors in missing input

6.2 Evaluation Methods

The Desktop Editor and Remote Editor applications were evaluated by both new and experienced game designers. For experienced game designers, feedback was requested from current game designers as well as other member of the OAR development team. These users are very familiar with the existing Game Editor and the concept of OAR games and thus have a clearer picture of the pros and cons of the existing and proposed interface designs. These users are also the targeted users for the project and can thus provide an evaluation matched against their own needs as designers. For new game designers, feedback was solicited from a group of students of various academic backgrounds who have moderate to high computer experience. This group of users is closest to simulating new game designers and can provide a new perspective on using the editing toolkit and help identify flaws more related to usability than content.

- Experienced Game Designers

The new editor applications were evaluated by designers and developers at various stages throughout the development process. During the design phase and earlier stages of developing working prototypes, mockups were sketched out for the riskier parts of the interfaces and reviewed by members of the OAR development team. These informal walkthroughs were of great value in gathering more ideas, challenging any assumptions made, flushing out difficult design questions, and identifying any major flaws early on in development.

The majority of the evaluations were performed after the completion of the application prototypes. Both the Desktop Editor and Remote Editor were presented to the development team with a walkthrough of the features and of the overall interface. These presentations also served as Q&A sessions to explain design decisions and clarify more ambiguous parts of the interface. Usability problems and possible solutions were also identified and discussed during these presentations and incremental changes were made to the applications accordingly. Having been completed earlier, a play-testable prototype for the Remote Editor was also posted to a content management and discussion site which is shared between the OAR team and collaborators at University of Wisconsin Madison and Harvard University.

- New Game Designers

Testing sessions were set up with the group of new users. Since testing of the Remote Editor is more valuable if the users are familiar with the previous method of creating OAR games, these testing sessions were only centered on the Desktop Editor. The testing sessions were held with five new users on two separate occasions, spaced one week apart.

In the first testing session, users were given a briefing on the concept of OAR games and the steps involved in creating a game. The first task they were given was to set up a new game with a given title, list of roles and chapters, game map, and a single game object. The second task they were given was to populate their game with three additional game objects and introduction text for every role. The information to populate the game was given in the form of an Excel spreadsheet. The tasks were performed on both the existing Game Editor and the new Desktop editor in randomized order and were timed as the standard of measurement.

The goal of the first testing session was to test learnability, efficiency, and error on each of the editor applications. Learnability was measured based on how fast users were able to find and populate the necessary options to complete the task, given minimal instructions. Efficiency was also measured by how quickly users were able to complete the task of adding multiple game objects. Users were given instructions on where to find game options before starting the task if they had not already discovered them during the previous task. Having completed the tasks, users were asked to check over their work before saving and finishing their games. The game files were then compared against a "solution" game file using a SVN diff program in order to identify any missing information or typos made during data entry.

In the second testing session on week later, the same users were asked to complete the same first task from the previous session. The task was again performed on both editors and timed for measurement. The goal this second session was to test memorability of using each editor, having performed the same task one week prior.

6.3 Results & Findings

This section presents an overview of the feedback and results gathered from the evaluation review and testing sessions.

- Experienced Game Designers

The Desktop Editor received a mixed reception from designers and developers. The content tables were recognized as a useful visualization of game data but could benefit from better organization and additional features such as spell checking, in place typing, and direct copy-paste functionality. In general, designers found the interface a little overwhelming with the added tab pages and property panels, but appreciated the ability to quickly flip between different game options. Another frequent comment made was regarding more contrast being made between different parts of the interface such as emphasizing tabs and headers and calling attention to important game options such as content table toolbars.

The Remote Editor received very good reception and designers were excited about using the new editing tool. The game object moving mechanic required a bit of explanation and will most likely require more revision but overall, designers liked the organization of the application, the clean and simplistic interface, and the functionality that the application offers. However, since no new game implementations were in progress during the evaluation period, no feedback from field usage was obtained.

- New Game Designers

The following table lists the results from the user testing sessions of the current Game Editor and the new Desktop Editor.

Table 6.4 Desktop Editor User Testing Results.

Task	User 1	User 2	User 3	User 4	User 5	Average
Task 1 (learnability)	5 min (DE) 7 min (GE)	9 min 8 min	7 min 7 min	8 min 11 min	10 min 8 min	7.8 min 8.2 min
Task 2 (efficiency)	11 min 17 min	12 min 14 min	18 min 22 min	11 min 14 min	14 min 19 min	13.2 min 17.2 min
Task 2 (errors)	2 errors 5 errors	1 errors 6 errors	0 errors 0 errors	4 errors 3 errors	2 errors 3 errors	1.8 errors 3.4 errors
Task 1 (memorability)	3 min 4 min	5 min 4 min	2 min 4 min	5 min 5 min	3 min 4 min	3.6 min 4.2 min

- Learnability

While performing the first task of setting up a new OAR game, users completed the task more quickly on the Desktop Editor, on average. The new game setup wizard was very beneficial in expediting the majority of the set up, particularly with respect to the title, roles, chapters, and game map. In the cases of both editors, the slowest portion of the task was initially location how to add game objects and where to populate their information. This was especially the case for the Desktop Editor since the application opens on to the "Map" tab, but content such as description and dialog text is currently only editable from the "Object Info" tab.

- Efficiency

While performing the second task of populating the OAR game, users again completed the task more quickly on the Desktop Editor, on average. All users found the spreadsheet style content tables very useful for quickly accessing all game object information in one location, though a couple users took a little longer to locate how to change the content table according to chapter. Some of the users were a little frustrated with the restrictiveness of the dialog forms in the Game Editor, but also found the option to apply content to all roles and chapters more quickly than on the Desktop Editor since the buttons are explicitly labeled on the buttons as opposed to relying on iconic buttons to identify the feature.

- Errors

With regard to errors, users made fewer mistakes on the Desktop Editor, on average. The majority of the mistakes made while using the existing Game Editor were missing text for a particular role or chapter. Users liked the ability to scan the content tables to find holes and typos and found the Game Editor interface tedious to use for checking over their work. Typos were not as prevalent a mistake since users generally just copy-pasted the text from the provided Excel spreadsheet directly into the editor applications.

- Memorability

During the second session, the Desktop Editor again performed better in terms of average timing, but this was also true for the Game Editor. The reduction of time to complete the task was consistent for both editors, so there was no large disparity between which interface was more memorable to use than the other.

Overall, the majority of users preferred the Desktop Editor over the existing Game Editor for completing the given tasks. Many of the usability problems identified from the testing sessions were similar to those noticed by game designers: visual complexity and low contrast between different parts of the interface. Though the tool tips were helpful in divulging the purpose of many features, many users suggested utilizing more text labels for buttons and features rather than relying on tool tips and iconic buttons.

7. Conclusion

7.1 Future Extensions

With any project, there is always a great disparity between how polished and how feature complete a developer wants a product to be and what time and resources allow. The following is a list of areas of improvement for the new editor applications and features that would be beneficial for the applications to support in the future.

- Flexible Map Tool

  The current Map Tool limits the game designer to capturing fixed size map images. This makes acquiring accurate game maps very difficult and designers are often forced to use maps that are too big or too small for the purposes of their implementation or attempt to create a map and position it manually. The Map Tool could be improved by allowing designers to select any region of a map and capture the image and coordinates accordingly. The image can then be resized by the Map Tool or by the Game Engine at run time.

- In-application Help Menu

  Providing quick and easy access to documentation via in-application help menus is a nice convenience for any level of users. Help menus are also a common feature for the majority of applications. The application documentation for the Desktop Editor and Remote Editor still reside largely outside the context of the applications and could be compiled into a help menu format. The challenge is then keeping all the information up to date.

- Excel Importing and Exporting

  As previously mentioned, many game designers use word formatting applications such as Microsoft Excel and Microsoft Word to plan out game content. If the OAR Game Editor supported importing and exporting of a standardized binary or text file format, designers would be able to completely bypass the process of manually porting text between spreadsheets and the editor application. The ability to export game content from the Game Editor back to a document format would be useful for designers who need to share content with other designers or collaborators and for keeping game data in the editor in sync with game content files.

- Timeline View

  The concept of a Timeline view is a way of visualizing game data according to game flow. Game flow in an OAR game is governed by a number of factors including game chapters, game object visibility, and object to object relationships such as triggering, anti-triggering, and containment. This type of visualization would help designers plan out the intended progress of players and more easily verify that all game flow related options are properly specified.

- In-application Preview

  Providing in-application previews of game object content such as text and image sizing would remove the need for the deploy-game-to-device and verify step, thus saving designers a lot of time and effort when testing games. Previews could be implemented by embedding a handheld device emulator into the Game Editor or by creating static handheld screens with overlaying text and images. The difficulty of the first method depends on the availability of an emulator that can be embedded within or used in parallel with the Game Editor. The second method would be easier to implement but more difficult to main and ensure perfect accuracy since sizing can vary with screen resolution.

7.2 Reflections

Though the Remote Editor has yet to be fully tested in practice, it has a lot of potential to become a very integral part of the OAR game creation process. By enabling on-site real-time game modification, the tool opens up more possibilities for on-location activities aside from designers gathering and testing game data. One very promising and immediate example is the investigation into students creating their own OAR games.

Whether the Desktop Editor stands to replace the existing Game Editor is still unclear, given that each version of the application still has its own benefits and limitations. Like any application interface, the new features and the Desktop Editor as a whole will require further evaluations and revisions to achieve a clean implementation. The development of the Desktop Editor has been an interesting experiment and a valuable exploration of new tools and methods of visualizing and working with game data which would be useful elements to include in any future iterations of the Game Editor application.

As the scope of OAR games continues to grow, the Game Editor toolkit will also have to evolve to accommodate all game functionality. Thus, an editing tool should not strive to find a catch-all solution for managing the ever-growing complexity, but rather remain flexible enough to support unforeseen changes. It is a tricky balance trying to support all possible features and functionality while trying to maintain a simple and easily learnable interface. As discovered during this project, solutions will often add their own set of complexities to the pool. Further iterations of revisions and evaluations will help achieve this balance and will help progress the Game Editor to its fullest potential.

8. References

- Accot, J. and Zhai, S. “Beyond Fitts’ law: models for trajectory-based HCI tasks.” Proceeding of the SIGCHI conference on Human factors in computing systems. (1997): 295-302.

- Bailey, J.E. and Pearson, S.W. “Development of a tool for measuring and analyzing computer user satisfaction.” Management Science. 29.5 (1983): 530-545.

- Billinghurst, M., Haller, M., and Thomas, B.H. Emerging Technologies of Augment Reality: Interfaces and Design. Hershey: Idea Group Publishing, 2007.

- Hackos, J.T. and Redish, J.C. User and Task Analysis for Interface Design. John Wiley & Sons, Inc., 1997.

- Hersh, H. and Rubinstein, R. The Human Factor. Bedford: Digital Press, 1984.

- James, G. “Book 4 – Coding.” The Tao of Programming. Santa Monica: Info Books, 1987.

- Klopfer, E. and Squire, K. “Environmental Detectives – The Development of an Augmented Reality Platform for Environmental Simulations.” Educational Technology Research and Development. (2005)

- Milgram, P. and Kishino, F. “A Taxonomy of Mixed Reality Visual Displays.” IEICE Transactions on Information Systems. E77-D(12) (1994): 1321-1329.

- “The Model-View-Controller Design Pattern” The Cocoa Fundamentals Guide. 22 July 2008. Apple Computer, Inc., 2006. <http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/chapter_5_section_4.html>

- Nielsen, J. “Usability 101: Introduction to Usability.” 23 August 2003. useit.com. 15 July 2008. < http://www.useit.com/alertbox/20030825.html>.

- Norman, D. Design of Everyday Things. New York: Basic Books (Perseus), 1988.

- Norton, M. “Augmented Reality Game Pilot Final Report.” (2007) Unpublished manuscript, Massachusetts Institute of Technology, Cambridge, MA.

- Pape, C. “Games Atelier – A Challenge for Collaborative Experience.” (2008) Unpublished manuscript, WAAG Society Amsterdam.

- Reason, J. Human Error. Cambridge University Press, 1990.

- Stenton, S. P., et al. “Mediascapes: Context-Aware Multimedia Experiences.” IEEE Multimedia. 14.3 (2007): 98-105.

- “What is Usability?” Usability.gov.15 July 2008. U.S. Department of Health & Human Servies. <http://www.usability.gov/basics/whatusa.html>.

A. Game Designer Survey & Interview Questions

The following list of questions was given to game designers as part of the user and tasks analyses.

- What type of computers do you normally use? (i.e. desktop, laptop, Apple, Windows, OS Version, etc.)
- What type of computers (machine specs) do you use to run the Editor?
- Do you have experience using any other editing applications? Which ones? (i.e. Word, Excel, Photoshop, etc.)
- How familiar are you with playing OutdoorAR games? Creating OutdoorAR games?
- How long have you been using the AR Game Editor?
- How many people work on designing each game?
- What is the time scope of each AR project? How many iterations does each game through?
- Please describe the steps that are typically performed when creating (designing AND building) an AR game from start (receiving curriculum/criteria for game) to finish (deploying finished game). The more detailed response the better.
- What step(s) require the most effort/work? Again, the more specific, the better.
- How are tasks divided up among the designers?
- What would you consider the biggest weaknesses/annoyances of the current editor? (not a wish list, just design flaws)
- If you could improve one thing about the existing Game Editor application, what would it be?

B. User Testing Briefing & Tasks

The following briefing and tasks were given to users during the evaluation testing sessions.

Briefing

Thank you for participating in this study! Today you will be creating your own Outdoor Augmented Reality game. But before you begin, here is a little bit of background information about what these games are exactly:

Outdoor Augmented Reality (OAR) games are games played at a specific geographic location using handheld devices with location tracking capabilities. You can think of OAR games as a mix between role-playing games and scavenger hunts. Players assume a certain character role and must explore the area to gather clues from virtual game objects displayed on the handheld device. These game objects are triggered by walking into its hotspot and can provide the player with information as text, audio, video, pictures, and other types of media. However, before you can start playing a game, you need to create one.

The goal of these testing sessions is to assess the interface of the applications, not you as a user. If you have any questions or problems while completing tasks please do not hesitate to ask for help.

Testing Session Task 1

Using the specified editor application, create a new game and populate it with the information provided in the table. For clarification, a description for each of the game settings is included in the table.

Game Setup Information

Info Type	Description	Value
Game Title	*title of the game*	MIT Rat Race
Player Roles	*name of selectable player roles*	Grumpy Grad Student
		Unwearied Undergrad
Game Chapters	*name and lengths (minutes) of game time segments*	Fall Term (20 min)
		Spring Term (15 min)
Game Map Image	*image file of game map*	mitmap.jpeg

Game Object 1

Name: Tim the Beaver
Type: NPC
Image: tim.jpeg
Description: MIT's loveable mascot. Tim is a very sociable creature and can be spotted giving hugs, taking pictures, and hanging out with students at most school functions.

Game Object 1 Dialog Information

Role	Fall Term	Spring Term
Grumpy Grad Student	Why the long face? Guess that research isn't starting out so smoothly for you huh? Hope this PhD comic strip cheers up your day!	They took away your funding?! Looks like info session pizza dinners for the next couple months...
	phdComic.html	
Unwearied Undergrad	How did that physics exam go?	Oh hey wait up! I think you dropped this...
	Are you checking out the hockey game tonight? The Tech is taking on the toughest game of the season! See you there!	dropform.jpeg

Testing Session Task 2

Using the specified editor application, add additional game objects to the game you have created. Populate the new game objects with the information provided in the table.

Game Introduction

Role	Page 1	Page 2	Page 3
Grumpy Grad Student	Welcome to MIT Orientation! Your mission, should you choose to accept it, is to survive the coming year at MIT.	If you make it through your mission, you will be duly rewarded with a shiny new Brass Rat. If you fail... well, there's always next year.	In the coming year your strength and stamina will be put to the test. Endless psets, grueling exams, restless nights... best of luck and welcome to the Institute.
Unwearied Undergrad	*same as above*	*same as above*	*same as above*

Game Object 2

Name: Heavy Textbook
Type: Item
Image: book.jpeg
Description: MIT is some back breaking work, literally. Better hit the books for that coming exam… before the books hit you!

Game Object 2 Dialog Information

Role	Fall Term	Spring Term
Grumpy Grad Student	Welcome to advanced quantum computational philosophy.	Congratulations, you have received funding for the semester as a Teaching Assistant. Read through this textbook and prepare the semester's worth of recitation material by next week.
Unwearied Undergrad	Welcome to 6.003 Signals and Systems.	gizmoball.avi
	Your first problem set is due tomorrow.	

Game Object 3

Name: Student Center
Type: Gate
Image: studcenter.jpeg
Description: Also known as Stratton Center, the student center is the central hub of MIT. Inside you will find a number of food vendors, the campus convenience store, and the largest Athena cluster on campus.

Game Object 3 Containment Information

Role	Fall Term	Spring Term
Grumpy Grad Student	*Contains:* nothing	*Contains:* nothing
Unwearied Undergrad	*Contains:* Heavy Textbook (code: 1861)	*Contains:* Tim the Beaver (code: 314159)

Game Object 4

Name: Ice Cream Study Break
Type: Checkpoint
Image: icecream.jpeg
Description: Feeling stressed? Take a little time off for a cold, frosty one. What's YOUR flavor?

Game Object 4 Code Information

Role	Fall Term	Spring Term
Grumpy Grad Student	*Code:* IHTFP	*Code:* Beaver Fever
Unwearied Undergrad	*Code:* ILTFP	*Code:* Punt and Tool

Wrap-up Task

Congratulations! You have successfully created an Outdoor Augmented Reality game. Before you finish, please go back and check over your work for any missing information or typos. A clean game implementation is always reviewed several times before being put to action.

Post-Mortem

- Which application interface did you prefer for Task 1?
- Which application interface did you prefer for Task 2?
- Which application interface did you prefer to use overall?
- Was there anything in either application that you really liked or disliked?
- What would you have changed in either of the interfaces (think mechanics and flow)?