

**Intuitive Fully Integrated Platform for
Designing Interactive Objects in Quest Atlantis**

by

David Lam

Submitted to the Department of Electrical Engineering and Computer Science in

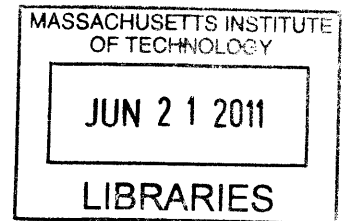
Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 2011
[June 2011]

Copyright 2011 David Lam. All rights reserved.



ARCHIVES

The author hereby grants to M.I.T. permission to reproduce and
to distribute publicly paper and electronic copies of this thesis document in whole and in part to any
medium now known or hereafter created.

Author _____
Department of Electrical Engineering and Computer Science
May 16, 2011

Certified by _____
Eric Klopfer
Associate Professor
Thesis Supervisor

Accepted by _____
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Intuitive Fully Integrated Platform for
Designing Interactive Objects in Quest Atlantis

by
David Lam

Submitted to the
Department of Electrical Engineering and Computer Science

May 16, 2011

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

ABSTRACT

Quest Atlantis is a 3D multi-user narrative game used as a teaching tool for children from ages 9 to 16. It has become highly successful and used by over 15,000 4th – 8th graders worldwide. Building upon this successful Quest Atlantis project, the designers of Quest Atlantis want to develop a narrative-based programming environment where users are able to design their own objects in the virtual world, script certain sequences of animations and behaviors into them, and share these narratives with others. Through this, users of this system will be able to learn fundamental computer science concepts and skills. Previous work has been done to build a simple platform that enables users to accomplish this. However, this previous platform was immature and lacked many of the important needed features. As a result, using this previous platform as the groundwork, I designed a fully integrated platform that is not only easy to use, but also contains all the features a user would need.

Thesis Supervisor: Eric Klopfer
Title: Associate Professor

Acknowledgements

I would like to thank Professor Eric Klopfer for giving me the opportunity to pursue this project to make this thesis possible. I would like to thank Daniel Wendel for his advice, guidance, and feedback throughout the course of the project. I would like to thank Adam Ingram-Goble for his help and feedback with the programming and implementation details of the project as well as his help in defining the focus of the project. Finally, I would like to thank the Scheller Teacher Education program and the designers of Quest Atlantis at Indiana University for all their help in making this project possible.

Contents

1	Introduction	9
2	Active Worlds Platform	
	2.1 Active Worlds Scripting Language	11
	2.2 Active Worlds SDK	12
3	Previous Work	
	3.1 Web Editor Interface for Creation of Objects	13
	3.2 Scripting of Objects using Flashblocks	14
	3.3 The Quest Atlantis Bot	16
	3.4 Database Tables	17
4	Initial Setup	
	4.1 Environment Setup	20
	4.2 Source Code Repository	21
	4.3 Migration from XVM Server to Scripts.MIT.EDU Server	22
5	Addition of New Commands	
	5.1 Say Command	23
	5.2 Play_Sound Command	24
	5.3 Walk Command	24
	5.4 Transport Command	24
	5.5 Transform Command	25
	5.6 Scale Command	25
	5.7 Media Command	26
6	Web Interface to Modify State Values of Objects	
	6.1 Index Listing of Created Objects	27
	6.2 Basic Object Info Modification	28
	6.3 Object Behavior Modification	30
	6.4 Object Position Modification	31
7	User Permissions	
	7.1 Database Setup	34
	7.2 Integration into Platform	35
8	Scriptblocks	
	8.1 Drawbacks of Flashblocks System	37
	8.2 Implementation of Scriptblocks	38
	8.3 Integrating Scriptblocks Into The Platform	39
9	Improvement of the QA Bot	
	9.1 Drawbacks of Current Quest Atlantis Bot	41
	9.2 Modification of Startup/Shutdown Process	42

9.3 Consistency Check Between Database and QA World	43
10 Admin Interface	
10.1 Creation of New Object Models	45
10.2 Creation of New Object Commands	46
10.3 Ability to Select Which Commands Each Model Can Have	50
10.4 Admin Home	53
11 Conclusion	54
12 References	56
Appendix	
A Setup	58
B Source Code – Quest Atlantis Bot	
B.1 bot_db.rb	59
B.2 object_editor.rb	60
C Source Code – Web-App Ruby Controllers	
C.1 admin_controller.rb	71
C.2 objects_controller.rb	73
D Source Code – Web-App Ruby Models	
D.1 animation.rb	78
D.2 owned_object.rb	78
D.3 palette_object.rb	78
D.4 project.rb	78
D.5 q_action.rb	78
E Source Code – Web-App Ruby Admin Views	
E.1 command.html.erb	79
E.2 listing.html.erb	80
E.3 model.html.erb	81
F Source Code – Web-App Ruby Object Views	
F.1 edit.html.erb	83
F.2 listing.html.erb	83
F.3 scriptblocks.html.erb	84
F.4 tweak.html.erb	86
G Source Code – Javascript	
G.1 numeric-stepper.js	89
G.2 ScriptblocksXML.js	91

1 Introduction

The use of technology as an education tool has been an extremely popular topic over the last few years. With the advancement in technology, researchers and scientists have been able to come up with new tools for learning. One of these tools is to use video games. Research has shown that there is educational potential in video games because they can help students discern and process information faster as well as increase the rate and accuracy of their reasoning skills¹. Quest Atlantis, an international learning and teaching project, is an example of one of these games.

Quest Atlantis is a 3D multi-user narrative game used as a teaching tool for children from ages 9 to 16. In this game, users set off on a variety of journeys that focus on a particular school subject. Throughout each journey, users are able to engage in a series of short educational tasks known as Quests, talk with other users, and build a virtual persona. Quest Atlantis has become highly successful and used by over 15,000 4th - 8th graders worldwide. It has played a critical role in standardized test gains, dozens of research publications, the creation of theoretical frameworks, the development of new media literacies, and the scaffolding of transformative experiences that involve both game-based and real-world narratives¹.

Building upon this successful Quest Atlantis project, the designers of Quest Atlantis want to develop a narrative-based programming environment where users are able to design their own objects in the virtual world, script certain sequences of animations and behaviors into them, and share these narratives with other users. Through this, the users of this system will be able to learn fundamental computer science programming concepts and skills. Unfortunately, the current Quest Atlantis system lacks an intuitive interface that would allow

¹ Barab, Sasha, "Transactive Narrative Art Proposal", Indiana University

users to perform all these needed actions. As a result, my role is to work with the designers of Quest Atlantis to build an intuitive platform that would allow users to design and script their own interactive objects so that they are able to produce their own interactive narrative.

This thesis begins with an examination of the technology that Quest Atlantis is built on as well as the previous work that has been done in developing this platform. The paper then proceeds to describe the development work that I did in building this platform.

2 Active Worlds Platform

Quest Atlantis is built using the Active Worlds virtual reality platform. The Active Worlds virtual reality platform contains two main methods to allow designers to build their own games on the platform: an SDK for building applications that function within the Active Worlds virtual environment and an object scripting language that allows designers to add animations and behaviors into objects. This section details these two Active Worlds technology.

2.1 Active Worlds Scripting Language

Each object in Quest Atlantis has an *action* state field that can be populated with statements using the Active Worlds Scripting Language. These statements will dictate the object's animation and behavior under certain conditions. The Active Worlds Scripting Language is used in the following way:

```
- trigger1 command1 param1 param2 ..., command2 param1 param2 ...;  
trigger2 command1 param1 param2 ..., ;
```

As we can see, any action script will begin with a trigger statement. All the commands that follow a trigger belong to the trigger and all the parameters that follow a command belong to the command. Each trigger is separated by “;” and each command is separated by a “,”. A trigger represents the condition in which its respective commands will be executed. For example, the *activate* trigger will execute its respective commands when the object is clicked².

Unfortunately, one of the drawbacks with using this Scripting Language is that it does not actually change the state of the object. For example, if we use the *move* command to move a given object 10 units forward, the object will move back to its original position after it

² Active Worlds Object Scripting. <http://wiki.activeworlds.com/index.php?title=Object_Scripting>

performs the action instead of staying in the new position. Another drawback with using Active Worlds Scripting Language is that the language does not allow us to chain commands. For example, assume that we have an object that has a script in the following format:

```
- activate command1 param1 command2 param2 command3 param3;
```

When we click on this object, instead of executing command1, command2, and command3 one by one in that sequence, it actually executes command1, command2, and command3 simultaneously. These drawbacks show the limitations in the Active Worlds Scripting Language.

2.2 Active Worlds SDK

The Active Worlds SDK is a C/C++ library that provides an easy way for developers to create applications that function within the Active Worlds virtual environment. The most common type of application that is built using the SDK is a bot. A bot is typically an avatar that lives in the virtual world, but is driven by a computer program instead of a human being. Using this SDK, we are able to manipulate objects in Quest Atlantis³. For example, the SDK allows us to permanently change the state of an object. As a result, using the SDK, we are able to permanently change the location of an object, something that could not be done using the Active Worlds Scripting Language.

Unfortunately, while we are able to use the SDK to permanently change the state of an object, we are not able to use the SDK to add animations and behaviors into an object. As a result, we need to use both the Active Worlds Scripting Language and the SDK to create animations and behaviors that are able to change the state of an object.

³ Active Worlds SDK. <<http://wiki.activeworlds.com/index.php?title=SDK>>

3 Previous Work

As discussed in section 2, Quest Atlantis is built using the Active Worlds virtual reality platform. Using a combination of Active Worlds scripting language and the SDK, previous work has been done to build a simple, relatively immature platform that allows users to design and script their own interactive objects so that they are able to produce their own interactive narrative. This section briefly details the previous work that has been done in building this platform⁴.

3.1 Web Editor Interface for Creation of Objects

One key aspect of this platform is the ability to allow users to create and add interactive objects into Quest Atlantis. Previous work has already been done to build a Ruby-on-Rails web application that allows for this functionality. While the back-end of this web application is built using Ruby-on-Rails, the front-end is built using a combination of HTML, CSS, and JavaScript.

Essentially, this web application contains a 2-dimensional grid that represents the coordinates in Quest Atlantis. On the right hand side is a panel consisting of the types of interactive objects that a user is able to create and add into Quest Atlantis. To

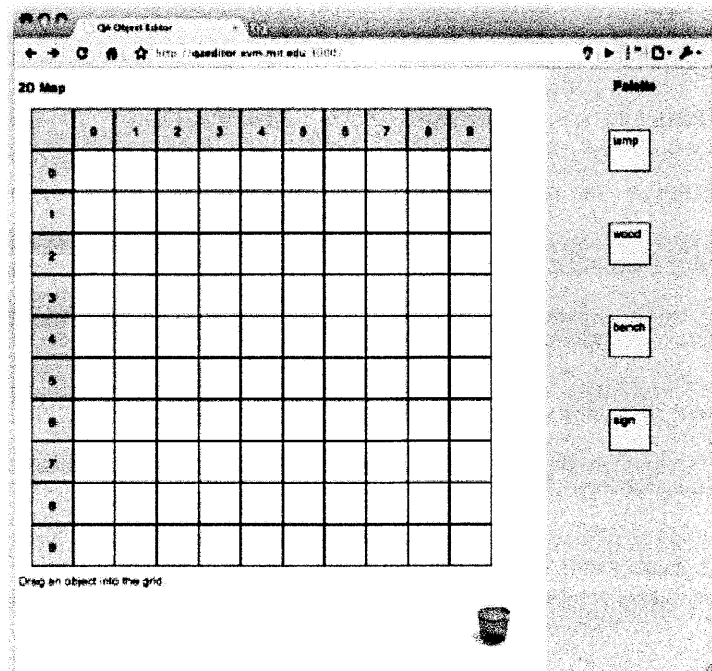


Figure 1: Web Interface that allows users to add their own interactive objects into Quest Atlantis

⁴ Irizarry, Angel, “Intuitive Interface for Object Interactivity and Storytelling for Quest Atlantis”, Master’s Thesis, Massachusetts Institute of Technology, 2010

add a particular object into Quest Atlantis, all the user has to do is drag that particular object from the right hand panel into the designated coordinate in the 2-dimensional grid.

The interactive objects that are available on the right hand panel of the Web application is determined by the Palette_Objects table of the database. This table is pre-populated with the information of all the available interactive objects that a user is able to create in Quest Atlantis.

The web application communicates with Quest Atlantis through the use of the database. When a user creates and adds an interactive object on the web application's 2-dimensional grid, an entry is added into the Q_Actions table of the database. A bot, discussed in section 3.3, running on a remote server will then read these entries from the Q_Actions table of the database and perform the necessary actions to add the specified interactive objects into the Quest Atlantis virtual world. Once the bot has created and added an object into the world, an entry containing information regarding this object gets added into the Owned_Objects table of the database. As a result, this Owned_Objects table will contain an entry for every object that is currently instantiated by users in the Quest Atlantis world.

3.2 Scripting of Objects using Flashblocks

Another key aspect of this platform is the ability to allow users to add sequences of animations and behaviors into the interactive objects that they have created in Quest Atlantis. Previous work has been done in this area to create a system that allows users to accomplish this through the use of Flashblocks. Flashblocks is an Adobe Flash implementation of the OpenBlocks block-programming framework, which was inspired by the work done on StarLogo TNG⁵. The use of Flashblocks makes the scripting of animations and behaviors into interactive objects in Quest Atlantis an extremely simple process. Users are not expected to

⁵ Roque, Ricarose Vallarata, "OpenBlocks : an extendable framework for graphical block programming systems", Master's Thesis, Massachusetts Institute of Technology, 2007

understand or learn any programming languages or concepts to use these Flashblocks. Essentially, all the user has to do is take these blocks and connect them together to form some sequence of animations and behaviors.

There are two main types of blocks that a user must use to create any sequence of animations and behaviors: trigger blocks and command blocks. Trigger blocks represent events that trigger series of animations and behaviors that are associated with each object. Command blocks represent the animations and behaviors that each object can have. Graphically, what separates these two types of blocks from each other are their shape and color. The shapes and colors of these two types of blocks enforce the constraints on how these blocks may be connected.

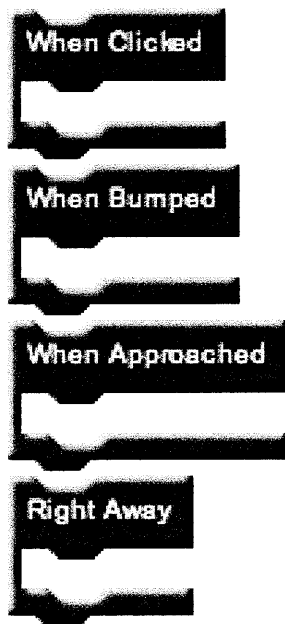


Figure 2: Trigger Blocks

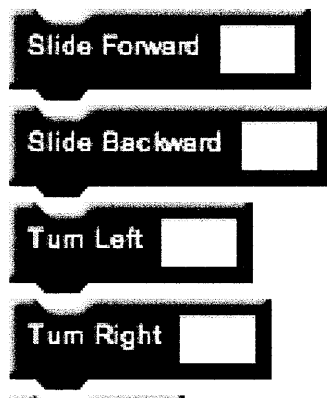


Figure 3: Command Blocks

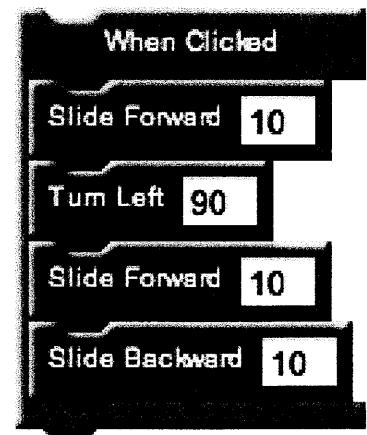


Figure 4: Trigger Block Connected with Command Blocks to Form A Sequence of Animations and Behaviors

3.3 The Quest Atlantis Bot

The Quest Atlantis Bot is the most important component and the heart of this platform. This bot runs on a separate server from Quest Atlantis and acts as the bridge between Quest Atlantis and the web application that allows users to create objects in the Quest Atlantis world as well as the bridge between Quest Atlantis and the Flashblocks system that allows users to add sequences of animations and behaviors into their objects. In addition, this bot is also responsible for handling event triggers as well as object animations and behaviors.

The bot facilitates communication from the web application and the Flashblocks to Quest Atlantis. The web application and Flashblocks each sends requests to the `Q_Actions` table in the database and the bot then takes each request in the `Q_Actions` table and processes it. There are three types of requests that the bot can process: create request, delete request, and script request. The *create* and *delete* requests are generated from the web application and are used to create and delete objects from Quest Atlantis. The *script* request is generated from the Flashblocks system and is used to add sequences of animations and behaviors into an object. The bot will periodically check the `Q_Actions` table to see if there are any new requests it needs to process. After processing any new requests, the bot will then delete those requests from the `Q_Actions` table. We can now see the critical role the bot plays in tying all the different components of the platform together.

Finally, the bot also plays a critical role in handling event triggers as well as object animations and behaviors. This is primarily where the use of the Active Worlds Scripting Language and Active Worlds SDK comes into play. The bot uses the scripting language to perform each animation and the SDK to listen to over 50 different types of events in Quest

Atlantis⁶. The main events that this bot is interested in are the events that are used to trigger series of animations and behaviors that are associated with each object. When an event does trigger a given object, the bot is responsible for carrying out the sequence of animations and behaviors that is associated with the object. The bot accomplishes this by spawning a new thread to execute the sequence of animations and behaviors. For each animation and behavior that the bot executes, the bot is also responsible for changing the object's state through the use of the SDK if the animation and behavior requires it.

3.4 Database Tables

There are three main tables in the database for this platform: `Palette_Objects`, `Owned_Objects`, and `Q_Actions`. The `Palette_Objects` table is pre-populated with all the different types of Quest Atlantis objects that a user is allowed to create. Each entry in this table consists of a model name, a human-readable name, and a description of the object. The `Owned_Objects` table contains all the objects that users have created in Quest Atlantis. Each entry in this table consists of an `AW_Object_ID`, the x, y, and z coordinates of the object, the exact orientation of the object, the action that the object is currently scripted with, and finally a `palette_object_id` that references the type of Quest Atlantis object the object belongs to.

The `Q_Actions` table is essentially a request queue that stores request from the web application and the Flashblocks system. For each request, this table is populated with key/value pairs that represent parameters for the request. As a result, a given entry in the `Q_Actions` table consists of an action name, an action id, a key, and a value. The action name represents what kind of request this entry belongs to. Since a given request has multiple entries in the table, the action id is a random number that is used to tie all the entries that belongs to the same request. The number of entries that a given request has is equal to the

⁶ Active Worlds Documentation. <http://wiki.activeworlds.com/index.php?title=Main_Page>

number of parameters that the request has. This type of setup makes the Q_Actions table extremely flexible. It is very easy to create new types of request and it allows a request to have a variable number of parameters.

id	action_name	action_id	key	value

Figure 5: Representation of Q_Actions Table

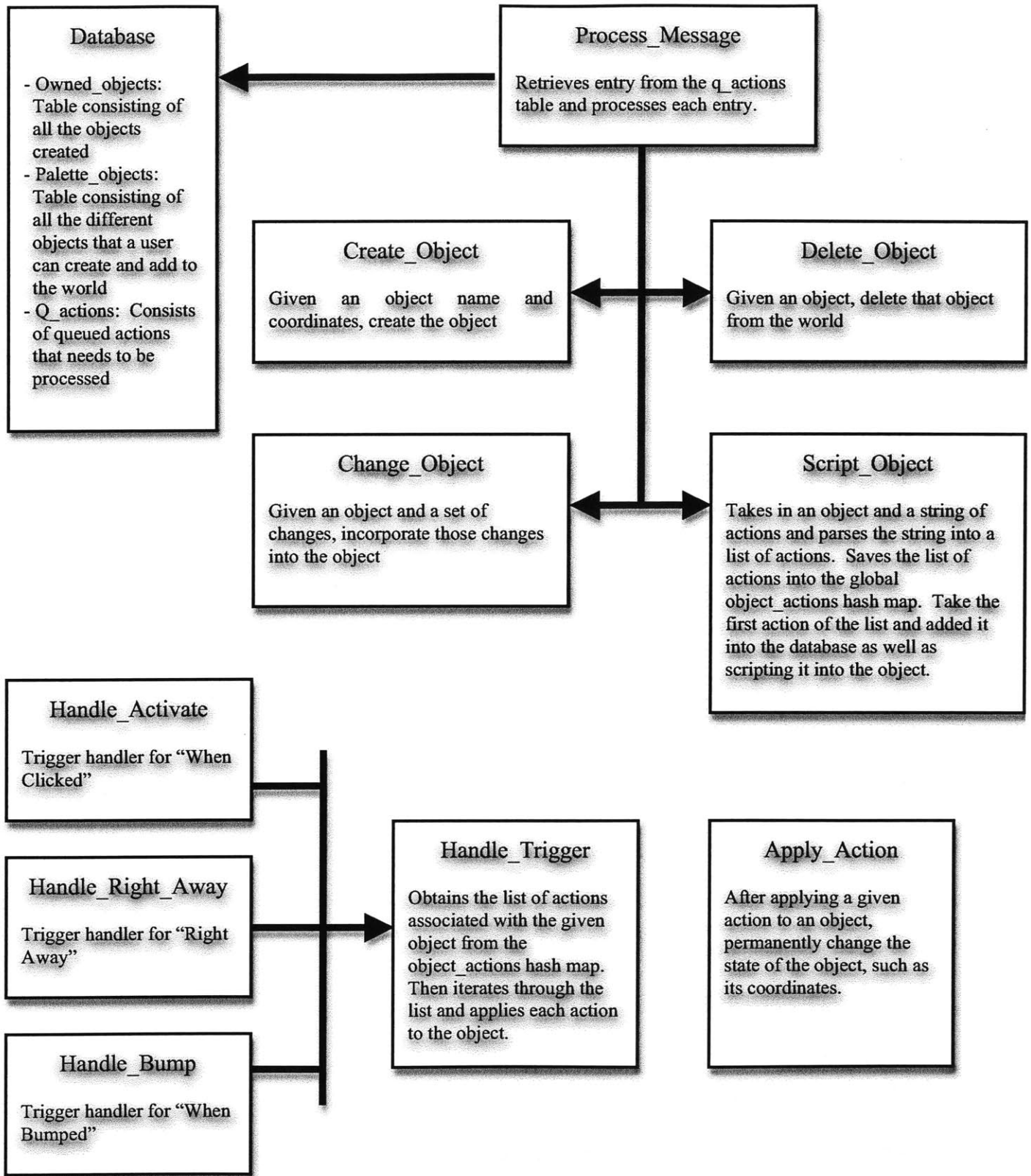


Figure 6: Flow Diagram of the Quest Atlantis Bot

4 Initial Setup

Previous work was done to build a simple platform that allowed users to design and script their own interactive objects so that they could produce their own interactive narrative. However, this platform was still immature and lacked a lot of important needed features. Therefore, using the previous work as a guideline, my role was to work with the designers of Quest Atlantis to build a fully integrated platform that was not only easy to use, but also contained all the features that a user would need. However, before doing so, the environment had to be set up and the source code for the bot had to be obtained from the previous platform. This section details all the initial setups that had to be done⁷.

4.1 Environment Setup

The Quest Atlantis Bot needed to run on a 32-bit Ubuntu operating system with the following installed:

- build-essential
- zlib1g-dev
- ruby
- rubygems
- ruby-dev
- libopenssl-ruby
- libmysql-ruby
- curl
- git-core.

All of these were installed using the Synaptic Package Manager. Next, the following RubyGems needed to be installed in this exact order:

⁷ Irizarry, Angel, "Intuitive Interface for Object Interactivity and Storytelling for Quest Atlantis", Master's Thesis, Massachusetts Institute of Technology, 2010

- `sudo gem install rails --version '=2.3.5'`
- `sudo gem install ffi --version '=0.3.5'`
- `sudo gem install activeworlds_ffi`

Finally, the Active Worlds SDK needed to be downloaded from the following location:

- <http://objects.activeworlds.com/downloads/awSDK77.tar.gz>.

A file named `libaw_sdk.42.so.77` then needed to be extracted from the downloaded file, renamed to `libaw_sdk.42.so`, and moved to the following location: `/usr/local/lib`. Since the `activeworlds_ffi` RubyGem searches the environment variable `LD_LIBRARY_PATH` for the Active Worlds SDK, we had to set this environment variable. This was done using the following terminal command:

- `export LD_LIBRARY_PATH = /usr/local/lib`

Fortunately, I was able to obtain a virtual machine image with the entire environment set up to the specifications provided above.

4.2 Source Code Repository

After having set up the Ubuntu environment to run the Quest Atlantis Bot, I needed to obtain the source code for the bot. This source code was obtained from the github repository by using the following command:

- `git clone git://github.com/angelman/QA-Object-Editor-Bot.git ObjectEditorBot`

The bot was then started by going to the `ObjectEditorBot` folder and running the following command:

- `ruby object_editor.rb 42982 qamit qadev`

4.3 Migration from XVM Server to Scripts.MIT.EDU Server

The Ruby-on-Rails web application that allowed users to create and add interactive objects into the Quest Atlantis world was originally hosted on the `xvm.mit.edu` server. Unfortunately, I did not have the necessary permissions to modify the specific instance of the `xvm.mit.edu` server that this web application was hosted on and there was not enough space on the `xvm.mit.edu` server at the time to run a new instance of the web application. As a result, I needed to migrate this web application on to a new server. I decided to use the `scripts.mit.edu` web hosting service because of its support for Ruby-on-Rails as well as its support for database hosting on `sql.mit.edu`. As a result, the Ruby-on-Rails web application is currently being hosted on `scripts.mit.edu` and the database is being hosted on `sql.mit.edu`.

5 Addition of New Commands

The available animations and behaviors that a user was able to add into objects using the previous simple platform included: slide forward, slide backward, turn left, and turn right. As we can see, this list of available actions was very limited and any user using this platform would want a richer list of actions. As a result, in building this platform, I added additional commands that a user is able to add on to an object. This section details the set of commands I added to the platform.

5.1 Say Command

The *say* command takes in a string and outputs the string on to the chat window in Quest Atlantis as part of the object's sequence of animations and behaviors. This command was created using Active Worlds Scripting Language *say* command⁸.

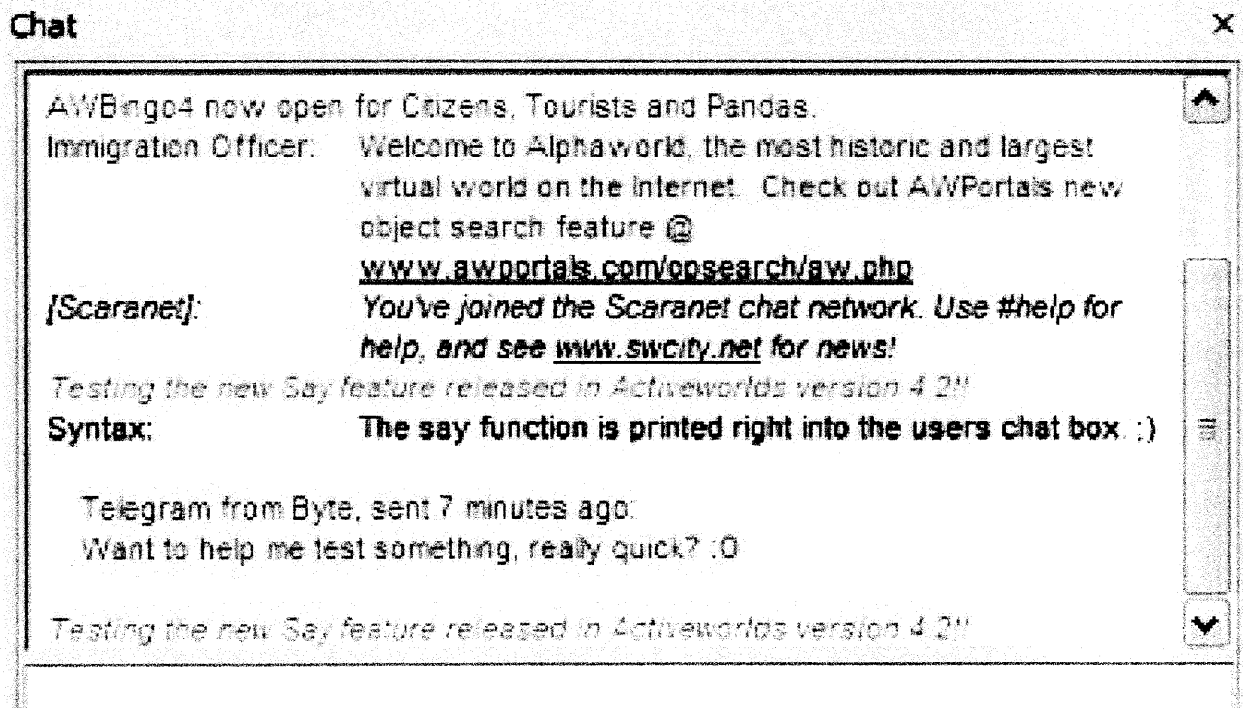


Figure 7: Example of an Output on to the Chat Screen by the *Say* Command

⁸ Active Worlds Say Command. <<http://wiki.activeworlds.com/index.php?title=Say>>

5.2 Play_Sound Command

The *play_sound* command takes in a URL to a sound file and allows the sound to be incorporated into an object and played as part of the object's sequence of animations and behaviors. The URL can be either the name of the sound file located in the "sounds" folder of the world's object path or it can be an absolute URL to a .wav, .mid, or .mp3 file anywhere on the web. When the sound file gets invoked, the sound will play over other sounds that are currently playing. This command was created using Active Worlds Scripting Language *noise* command⁹.

5.3 Walk Command

The *walk* command takes in a number and dictates how far a created character object should walk as part of its sequence of animations and behaviors. This command was created using a combination of Active Worlds Scripting Language *seq* command and *move* command. The *seq* command takes a specified animation sequence that is part of the object's path and applies it to an object¹⁰. In this case, the animation sequence that is used is the *walk* sequence. However, this *walk* animation sequence does not actually move the object. To move the object, we had to use the *move* command¹¹. Thus, by simultaneously using the *seq* and *move* command, we are able to create the walk command. The Active Worlds Script to do this is:
seq walk, move x y z.

5.4 Teleport Command

The *teleport* command takes in a location and teleports the user to the new location as part of the object's sequence of animations and behaviors. This *teleport* command can only

⁹ Active Worlds Noise Command. <<http://wiki.activeworlds.com/index.php?title=Noise>>

¹⁰ Active Worlds Seq Command. <[http://wiki.activeworlds.com/index.php?title=Seq_\(building_command\)](http://wiki.activeworlds.com/index.php?title=Seq_(building_command))>

¹¹ Active Worlds Move Command. <<http://wiki.activeworlds.com/index.php?title=Move>>

teleport the user to a new location that is in the same world. In other words, the *teleport* command cannot teleport the user to a different world. This command was created using the Active Worlds Scripting Language *warp* command¹².

Unfortunately, due to the platform's inability for commands to take in more than one argument, the *teleport* command had to be broken down into a *teleport_x* and a *teleport_z* command. As the names suggest, the *teleport_x* command sends the user to a new x location while keeping the z location constant and the *teleport_z* command sends the user to a new z location while keeping the x location constant.

5.5 Transform Command

The *transform* command takes in an object model name and transforms the object to this new object model as part of the object's sequence of animations and behaviors. If a non-valid object model is given as the input, then the *transform* command would not do anything. This *transform* command requires the manipulation of objects in the Quest Atlantis World which is something that cannot be done using Active Worlds Scripting Language. As a result, this command had to be created using Active Worlds SDK, specifically through the Quest Atlantis Bot.

5.6 Scale Command

The *scale* command takes in a scale factor and changes the size of the object by that scale factor as part of the object's sequence of animations and behaviors. For example, if the scale factor is 2.0, the command will make a given object twice as big. Similarly, if the scale factor is 1.0, the command would not do anything to the given object. This command was created using Active Worlds Scripting Language *scale* command¹³.

¹² Active Worlds Warp Command. <<http://wiki.activeworlds.com/index.php?title=Warp>>

¹³ Active Worlds Scale Command. <<http://wiki.activeworlds.com/index.php?title=Scale>>

5.7 Media Command

The *media* command takes in an URL to a media file and plays that media file as part of the object's sequence of animations and behaviors. Similar to the *play_sound* command, the URL can be either the path to a media file or it can be an absolute URL to a media file anywhere on the web. The types of media files that are supported are the same as the types of media files that are supported by the installed version of Windows Media Player. This command was created using Active Worlds Scripting Language *media* command¹⁴.

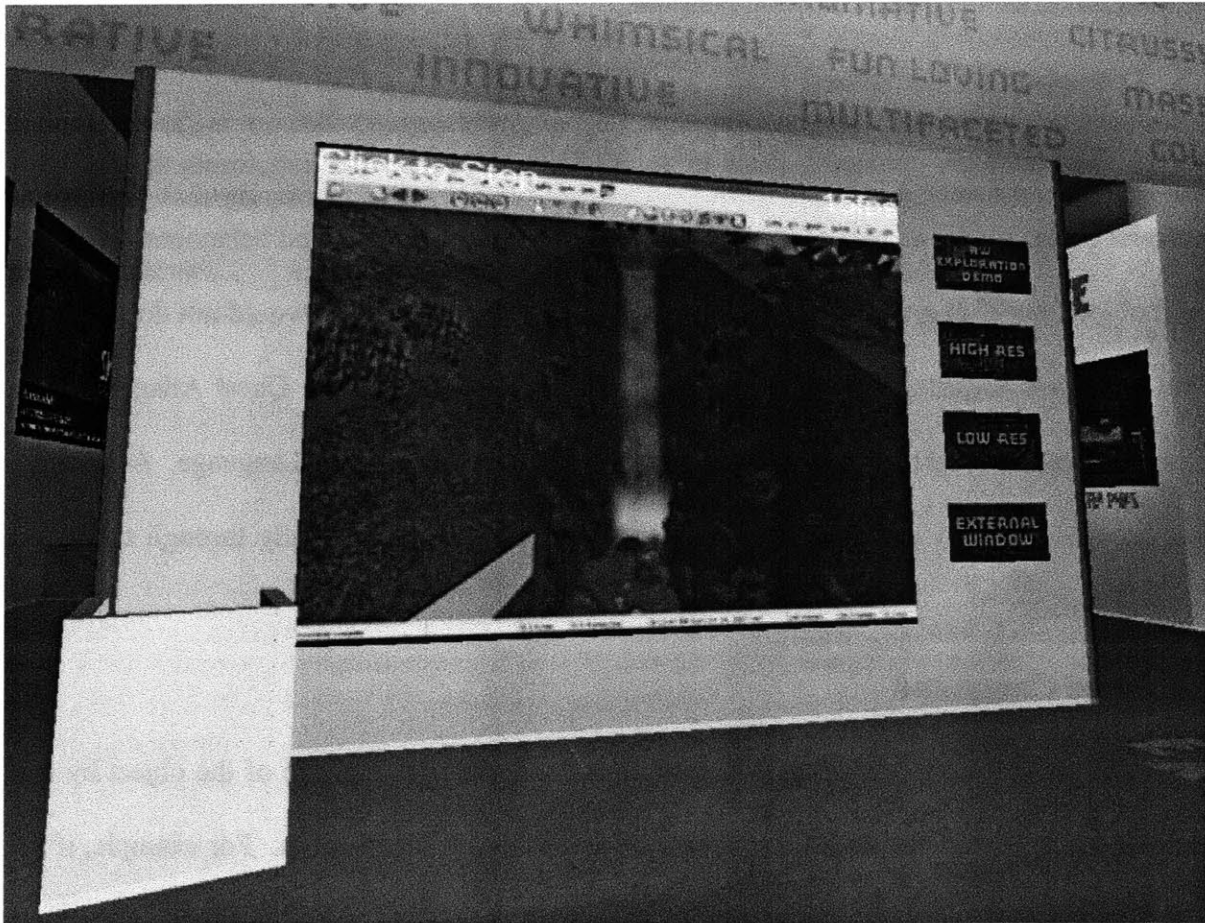


Figure 8: Example of a Streaming Media File Using the Media Command

¹⁴ Active Worlds Media Command. <<http://wiki.activeworlds.com/index.php?title=Media>>

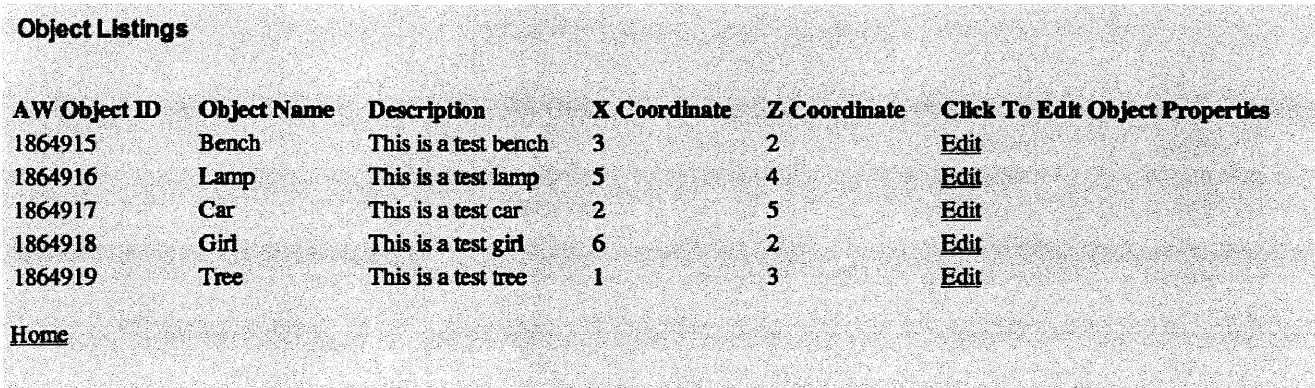
6 Web Interface to Modify State Values of Objects

Users in Quest Atlantis are able to modify the states, such as location and rotation, of any interactive objects that they have created in Quest Atlantis. To do so, all the user had to do was go to Quest Atlantis, find the object, and click on the object to open the sidebar interface containing the modifiable state fields for the given object. In addition, the Flashblocks system where users are able to script their own sequences of animations and behaviors into objects also had to be accessed through the sidebar of the Quest Atlantis game interface. While this type of setup worked, it was not ideal. Instead, we envision a fully integrated web interface where users should be able to accomplish all their desired tasks in designing their own interactive objects without having to go through Quest Atlantis or any other program. This section details the work that was done to accomplish this.

6.1 Index Listing of Created Objects

Before we were able to create this fully integrated web interface, we had to find a way for users to select which objects they want to modify without selecting the object through Quest Atlantis. As a result, as part of the Ruby-on-Rails web application where users are able to create and add their own interactive objects into the world, we added a link that leads to an index listing of all the interactive objects that have been created. In addition to the object name, for each object, this listing contains other information regarding the object as well as an edit link for the object. Now when a user wants to modify the state values of an object or script new animations and behaviors into an object, all the user has to do is click on the edit link of the desired object from this listing and a new web interface containing the modifiable state values and a link to the Flashblocks interface will open up. By using the new permissions checking logic as discussed in section 7, for any given user, this index listing of

objects is restricted to only contain objects that were created by the given user or those that were created by the given user's teammates.



Object Listings

AW Object ID	Object Name	Description	X Coordinate	Z Coordinate	Click To Edit Object Properties
1864915	Bench	This is a test bench	3	2	Edit
1864916	Lamp	This is a test lamp	5	4	Edit
1864917	Car	This is a test car	2	5	Edit
1864918	Girl	This is a test girl	6	2	Edit
1864919	Tree	This is a test tree	1	3	Edit

[Home](#)

Figure 9: The Index Listing of Created Objects page

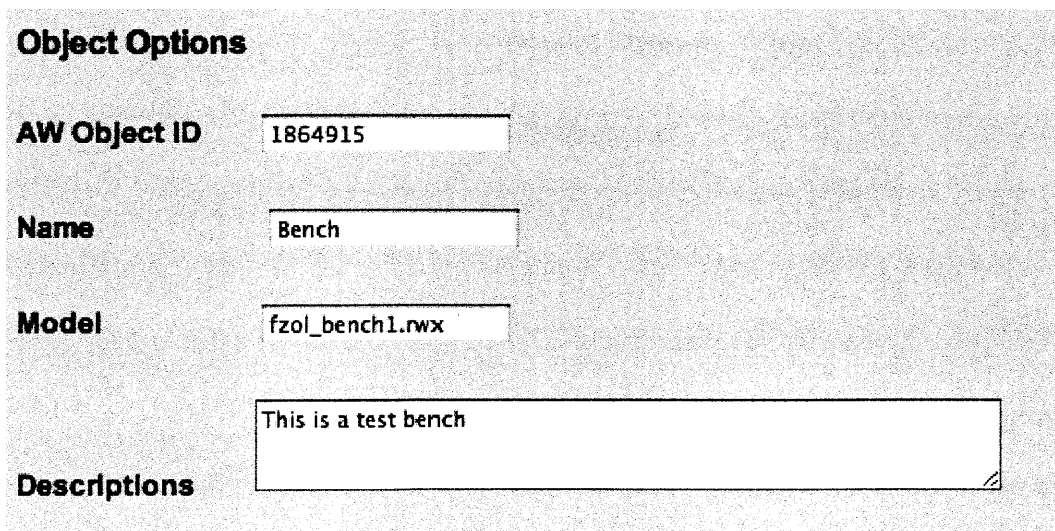
6.2 Basic Object State Modification

In creating this new web interface, we envision users being able to modify the *name*, *model name*, and *description* state fields of a given object. To do so, we first had to add two additional fields into the Owned_Objects table of the database. These two additional fields were the *name* field and the *description* field. We did not need to add a *model* field into the table because that field was already represented in the Palette_Objects table, which gets referenced by the Owned_Objects table.

The *name* field is not actually a valid field in Quest Atlantis' representation of an object. This field only exists in the web interface as a way to allow users to easily identify their objects by giving each object their own unique name. However, both the *model name* field and the *description* field are valid fields in Quest Atlantis' representation of an object. The *model name* field of an object dictates what kind of Quest Atlantis object the given object belongs to. The *description* field of an object is an informative piece of text that gives a

description of the object. As a result, changing the *model name* field or the *description* field of an object involves the manipulation of the state of the given object. To do this, we used the Active Worlds SDK, through the Quest Atlantis bot, to change these state values of the object. The SDK method to change the model name of an object is *aw_string_set(AW_OBJECT_MODEL, new_model_name)* and the SDK method to change the description of an object is *aw_string_set(AW_OBJECT_DESCRIPTION, new_description)*.

As discussed in section 3.3, the different components of this platform communicate with the Quest Atlantis bot mainly through the use of the Q_Actions table of the database. This was no different with this new web interface. To allow this new web interface to communicate with the bot, a new type of request known as a *change* request was created. Each *change* request added into the Q_Actions table consisted of two entries with the keys *model* and *description*. The values associated with the keys *model* and *description* are the new model name and the new description, respectively, of the given object. Once a *change* request gets added into the Q_Actions table, the bot will process this request and delete this request from the table.



The image shows a web form titled "Object Options" with four input fields. The first field is "AW Object ID" with the value "1864915". The second field is "Name" with the value "Bench". The third field is "Model" with the value "fzol_bench1.rwx". The fourth field is "Descriptions" with the value "This is a test bench".

Object Options	
AW Object ID	1864915
Name	Bench
Model	fzol_bench1.rwx
Descriptions	This is a test bench

Figure 10: Users can now modify the *name*, *model name*, and *description* field of an object

6.3 Object Behavior Modification

In addition to having the ability to modify the *name*, *model name*, and *description* fields of an object via this web interface, users should also have the ability to modify the *actions* field of a given object. The *actions* field is essentially the sequence of animations and behaviors that an object will execute under certain trigger events. Currently, the only way a user is able to script animation and behavior sequence into an object is through the Flashblocks system. However, in addition to this Flashblocks System, we also want users to have the option to modify an object's animation and behavior sequence through a textbox.

To accomplish this, we added an additional *actions* field into the Owned_Objects table of the database. This *actions* field now contains the string representation of the sequence of animations and behaviors for a given object. Since the Flashblocks System communicates with the Quest Atlantis bot by outputting a string representation of the scripted sequence of animations and behaviors into the Q_Actions table, we simply used this same string to represent this new *actions* field in the Owned_Objects table. The following is an example of this string:

```
- {When_Clicked[Slide_Forward(5), Turn_Left(90), Slide_Forward(5)]}
```

Basically, when the object scripted with the above string gets clicked on, it is supposed to move forward five units, rotate left ninety degrees, and move forward another five units.

Propagating changes in an object's animation and behavior sequence to Quest Atlantis again involved the use of the Quest Atlantis bot. Similar to all the other components of this platform, communicating changes in the *actions* field for an object was done through the Q_Actions table of the database. Each *actions* field change added into the Q_Actions table was considered as a *change* request entry with the key *action*. The value associated with the

key *action* is the string representation of the new animation and behavior sequence of the given object.

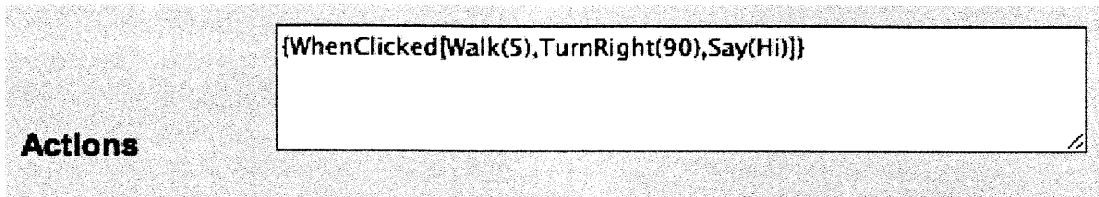


Figure 11: Users can now modify the *actions* field of an object

6.4 Object Position Modification

This new web interface should also allow users to have the ability to modify the *location* and *orientation* of an object in Quest Atlantis. Again, similar to all the other modifiable fields in this new web interface, these location and orientation changes gets communicated to the Quest Atlantis bot through the Q_Actions table of the database. Each *location* field change added into the Q_Actions table is consider as three *change* request entries with the keys *x*, *y*, and *z*. These three keys represented the 3D *x*, *y*, and *z* positions, respectively, in the Quest Atlantis world. Each rotation field change added to the Q_Actions table is considered as three *change* request entries with the keys *yaw*, *tilt*, and *roll*. These three keys represented the possible rotation orientation on the *y*, *x*, and *z* axis respectively. Once a *location* or *rotation change* request is inserted into the Q_Actions table, the bot will retrieve and process these request and change these states on the given object using Active Worlds SDK. The SDK methods to carry out these changes are:

- `aw_int_set(AW_OBJECT_X, new_x)`
- `aw_int_set(AW_OBJECT_Y, new_y)`
- `aw_int_set(AW_OBJECT_Z, new_z)`
- `aw_int_set(AW_OBJECT_YAW, new_yaw)`

- `aw_int_set(AW_OBJECT_TILT, new_tilt)`
- `aw_int_set(AW_OBJECT_ROLL, new_roll)`

Originally, we designed these modifiable *location* and *orientation* fields on this new web interface as standard html input text fields where users are able to enter any input of their choice. However, we found that this was not optimal since users were able to enter a non-valid location or orientation as inputs. Instead, we wanted to restrict users to only be able to enter valid inputs into these fields. As a result, we decided to make the *location* and *orientation* input fields so that each field had both an up and down arrow button. By having this, users are only able to modify the *location* and *orientation* field by incrementing the field value or decrementing the field value through the up/down arrow buttons. Users are limited by how high they can increment the field value and the same goes for how low they can decrement the field value. This was accomplished using a combination of html and jQuery, which is a powerful Javascript library¹⁵.

By using jQuery technology, we also made it so that when a user presses an up or down arrow button on any of the *location* and *orientation* fields for an object, that object in Quest Atlantis will receive the update in real-time without having the user press the save button on the web interface. For example, if a user presses an up arrow button on the *x location* field, the respective object in Quest Atlantis will move forward in the x direction by one unit in real-time. This allows users to get a better sense of exactly how the *location* and *orientation* changes they are applying to the object is affecting the object as well as how they should go about applying these changes.

¹⁵ Javascript Numeric Stepper. <<http://www.htmldrive.net/items/show/540/Javascript-numeric-stepper-with-inputbox.html>>

Object Options

AW Object ID

Name

Model

Descriptions

Actions

Location X (W-E): Y (Up-Down): Z (N-S):

Rotation Yaw (Y Axis): Tilt (X Axis): Roll (Z Axis):

[Animate Object!](#)

[Back to Object Listing](#)

Figure 12: New Web Interface To Modify State Values of Objects

7 User Permissions

One of the biggest flaws with the previous platform was that users were able to modify the interactive objects created by other users. As a result, from the user's standpoint, there was no sense of ownership of objects since any user was able to freely modify any interactive objects that were created in Quest Atlantis. As a result, we added the use of permissions into this new platform to limit users to only be allowed to modify certain objects. Essentially, users will only have permission to modify the interactive objects that they created and the interactive objects that their respective teammates created. This section details the work that was done to accomplish this.

7.1 Database Setup

Before we could begin adding the permissions checking logic into this platform, we first modified the database so that it could store the information necessary to perform this permissions checking. To do so, we added a *Projects* table into the database that had a one-to-many mapping with the *Owned_Objects* table and a many-to-many mapping with users of Quest Atlantis. In other words, all objects created in Quest Atlantis can belong to only one of the projects in the *Projects* table and all users in Quest Atlantis can belong to any of the projects in the *Projects* table. To accomplish this type of mapping between tables, a *project_id* field was added to the *Owned_Objects* table and a separate table containing *project_id/user_id* pairings was added.

Based on this database setup, a given user could modify a particular object only if that user belongs to the same project that the object belongs to. Since multiple users can belong to the same project, multiple users can have the ability to modify the same object. Similarly,

since a user can belong to multiple projects, a user could modify multiple objects where each object belongs to a different project.

7.2 Integration into Platform

After setting up the database, we added the permissions checking logic into the platform. Specifically, we modified the web editor application as well the Quest Atlantis bot. As discussed in section 3.1, the web editor application contains a 2-dimensional grid that displays the coordinates and the relative location of all the interactive objects that were created and added into Quest Atlantis. We modified this web editor application to take in a *project_id* variable and used this *project_id* to only display objects that belong to the project with this *project_id*. Additionally, since users of this platform use this web application to create and add their own objects into Quest Atlantis, we modified this web editor application so that any object created will belong to the project with this *project_id*.

As discussed in section 6.1, a new addition to the web editor application is an index listing of all the interactive objects that were created by a given user or that user's teammates. This was also accomplished by using this new permissions checking logic. Assuming that the given user and all of the user's teammate belong to the same project with some *project_id*, then we used this *project_id*, which gets passed in as an argument into the web editor application, to only display objects that belong to the project with this *project_id*.

Finally, we modified the Quest Atlantis bot as well as the communication layer between the bot and the web editor so that it could perform permissions checking on any object modification requests coming to the bot. As discussed in section 3.3 and 3.4, the web editor communicates with the bot mainly through the use of the Q_Actions table. For any modification request that the web editor adds to the Q_Actions table, we made it so that an

additional entry that belongs to this request gets added into the table with the key *project_id*. The value associated with this key is the *project_id* parameter that got passed in earlier as an argument to the web editor. In other words, each modification request that gets communicated from the web editor to the bot will contain an additional *project_id* parameter. The bot then checks to ensure that for each request, the interactive object that the request is for has the same *project_id* as that of the request.

8 Scriptblocks

As discussed in section 3.2, Flashblocks is an Adobe Flash implementation of the OpenBlocks block-programming framework that makes the scripting of animations and behaviors into interactive objects in Quest Atlantis an extremely simple process. All the user has to do is take these blocks and connect them together to form some sequence of animations and behaviors. Unfortunately, while the Flashblocks system accomplishes the task of providing an easy way for users to script animations and behaviors into interactive objects, the Flashblocks system does have its limitations. This section details the drawbacks of the Flashblocks system as well as a new block-programming framework that we decided to replace Flashblocks with in our platform.

8.1 Drawbacks of the Flashblocks System

One of the biggest limitations with the Flashblocks system is that it did not have the ability to save a snapshot of the blocks that users had assembled together. In other words, after a user had finished assembling together the blocks to form some sequence of animations and behaviors for a given object, the user was not able to go back and make a small change to that block structure. Instead, users had to re-assemble the Flashblocks every time they wanted to make some change to the animation and behavior sequence of an object. As we can see, this type of setup was very limited.

As a result, we decided to look into modifying the Flashblocks system to try to fix this limitation. Unfortunately, another limitation of the Flashblocks system was the lack of documentation on the system. This made it incredibly difficult to make any changes and improvements to the system. Because of this, we came to a quick realization that modifying the Flashblocks system to fix its limitations was not a feasible option.

We then decided to look into a new block-programming framework known as Scriptblocks, which was being built by researchers from the MIT Media Lab. After studying this new framework, we saw that Scriptblocks solved all the problems that we had with the Flashblocks system. Thus, we believed that the Scriptblocks system was a viable candidate to replace the Flashblocks system in our platform.

8.2 Implementation of Scriptblocks

Scriptblocks was built using Javascript and the Google Closure Tools. As a result, the Scriptblocks framework did not require any additional plug-ins to run and could easily be integrated into our platform. An important part of the Scriptblocks framework was an XML library that allows us to specify via an XML file, the different types of blocks our system needs. This made Scriptblocks extremely flexible as we were able to change the specifications of our blocks without the need to modify and re-compile the Scriptblocks source code.

Additionally, the Scriptblocks XML library allowed us to save to a XML file a snapshot of the workspace containing the blocks that we had assembled together. After outputting the XML file, the Scriptblocks XML library then had the ability to parse the XML file back in to reconstruct the entire block structure that we had assembled earlier. In other words, in the Scriptblocks framework, after we finished connecting the blocks together to form some block structure, we could go back to the block structure and make changes to it without having to re-assemble the entire block structure.

As we can see, this new Scriptblocks system is a great improvement compared to the Flashblocks system. It lacks the limitations that the Flashblocks system had and as a result, it solved our entire problem with the Flashblocks system.

8.3 Integrating Scriptblocks Into the Platform

To integrate the new Scriptblocks system into our platform, we had to make a few modifications on both the Scriptblocks system and our platform. As discussed in section 6.3, the Flashblocks System communicated with the Quest Atlantis bot by outputting a string representation of the scripted sequence of animations and behaviors into the Q_Actions table of that database. The following is an example of this string:

```
- {When_Clicked[Slide_Forward(5), Turn_Left(90), Slide_Forward(5)]}
```

If we want to integrate the Scriptblocks system into our platform without modifying the Quest Atlantis bot, then we would need to modify the Scriptblocks system to output a similar string representation of the scripted sequence of animations and behaviors. As a result, we added this functionality to the Scriptblocks system. After adding this functionality to the Scriptblocks system, we created the XML file detailing the block specifications for our platform. Essentially, there are three types of blocks in our system: trigger, command, and constant. Shown below is a table detailing the blocks we needed for our platform.

Trigger Blocks	Command Blocks	Constant Blocks
WhenClicked	SlideForward	1
WhenBumped	SlideBackward	5
WhenApproached	TurnLeft	10
RightAway	TurnRight	25
	Walk	"This is a test string"
	PlaySound	
	Say	
	Transport	
	Transform	
	Scale	
	Media	

Figure 13: Table Showing the Needed Blocks For the Scriptblocks System

After making these modifications to the Scriptblocks system, we had to make some modifications to our platform before we were able to completely integrate the new Scriptblocks system into the platform. Specifically, we added an additional *block_XML* field to the Owned_Objects table of the database. This field contained the XML of the saved snapshot of the blocks that a user had assembled together for the respective object. With this modification, the Scriptblocks system is now able to output the XML string of the saved snapshot into the database instead of outputting it to a file. Similarly, the Scriptblocks system is now able to reconstruct the entire block structure that was assembled for a given object by reading in that XML string from the database. After all these modifications, we were able to completely integrate the Scriptblocks system into the platform as well as removed the Flashblocks system from the platform.

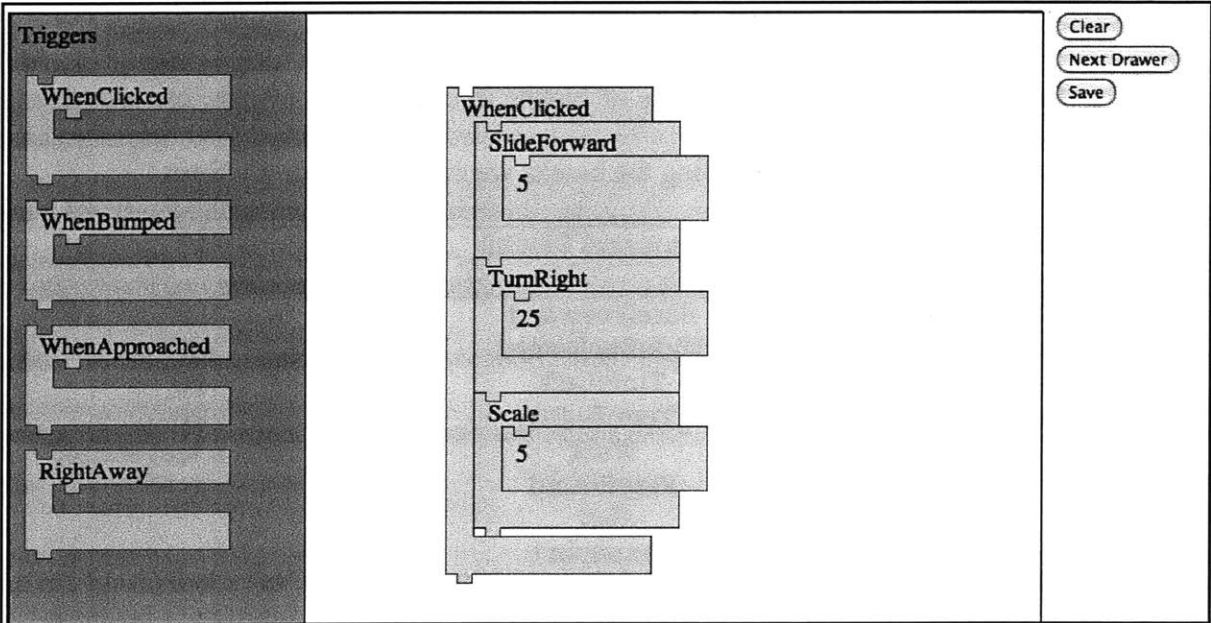


Figure 14: The Scriptblocks Workspace

9 Improvement of QA Bot

As discussed in section 3.3, the Quest Atlantis bot is the most important component and the heart of this platform. Running independently on a Linux server, the bot has three main responsibilities: 1) facilitating communication between the new fully-integrated web application and Quest Atlantis, 2) responding to event triggers in Quest Atlantis, and 3) handling object animations and behaviors. Unfortunately, the Quest Atlantis bot is still limited in certain respects and there are a lot of improvements that can be done to it. This section details the drawbacks of the current Quest Atlantis bot and the different improvements that were made to it.

9.1 Drawbacks of Current Quest Atlantis Bot

One of the biggest drawbacks with the current Quest Atlantis bot is that it did not persist the sequences of animations and behaviors for all created objects in Quest Atlantis into the database. Instead, all the animation and behavior information for the objects were saved in memory. As a result, when the Quest Atlantis bot was restarted, the sequences of animations and behaviors for all the objects got deleted and were lost forever. Since, the bots in practice will be restarted on a regular basis, this type of setup did not work.

Another drawback with the current Quest Atlantis bot is that during the startup process as well as the shutdown process of the bot, the bot deleted all the existing interactive objects that were created from the database. The bot accomplished this by going into the Owned_Objects table of the database and removing all the entries in that table. Not only did this delete all the hard work users put into creating their interactive objects, but this also created an inconsistency between Quest Atlantis and the database. While the bot may have deleted all the entries from the Owned_Objects table during its startup and shutdown process,

the bot never deleted the respective interactive objects from Quest Atlantis using the Active Worlds SDK. As a result, all the interactive objects that were created still remained in Quest Atlantis. However, since the bot no longer had knowledge of these objects after it had restarted, the bot was not be able to respond to any event triggers or execute the animations and behaviors that were associated with these objects. Again, since the bots will be restarted on a regular basis, this type of setup did not work.

According to the documentation detailing the design of the Quest Atlantis bot, the bot was designed this way because bots should only be able to manipulate and animate objects that they own¹⁶. Since there will be multiple bots running and reading from the same database, for any given object that gets created, the object belongs to the bot that instantiated the object and added it to Quest Atlantis. Therefore, when the bot exits or when a new bot starts up, the records of all owned objects must be cleared to prevent the new bot and any future bots from attempting to manipulate and animate objects it did not own.

9.2 Modification of Startup/Shutdown Process

According to the designers of Quest Atlantis, since there are multiple worlds in Quest Atlantis, they planned to have a single instance of the Quest Atlantis bot for each of these worlds. Because of this, instead of having the restriction that bots should only be able to manipulate and animate objects that they own, we modified this restriction to be: bots should only be able to manipulate and animate objects that belonged to the same Quest Atlantis world as the bot. By modifying this restriction, we now could modify the startup process and shutdown process of the bot so that the bot does not delete any of the created interactive objects from the database. To accomplish this, an additional *qa_world* field was added into the Owned_Objects table of the database. This new *qa_world* field contains the Quest

¹⁶ Irizarry, Angel, "Intuitive Interface for Object Interactivity and Storytelling for Quest Atlantis", Master's Thesis, Massachusetts Institute of Technology, 2010

Atlantis world that the respective object belongs to. Before manipulating or animating an object, the bot will use this *qa_world* field of the object to first check to ensure that the object and the bot belong to the same world. By doing this, the Quest Atlantis bots are now able to freely exit and restart without affecting any of the interactive objects that were created.

However, even after modifying the bot so that it does not delete any of the created interactive objects from the database during its startup and shutdown process, we still ran into the problem of losing the animation and behavior information for all the objects once the bot got restarted. Therefore, for each object, we had to persist the animation and behavior information into the database. Fortunately, as a side effect of the new functionality that allows users to modify an object's *action* field, an object's animation and behavior information was already persisted into the database. As discussed in section 6.3, the new *actions* field in the Owned_Objects table of the database contains the string representation of the sequence of animations and behaviors of the respective object. Using this *actions* field of the Owned_Objects table, we were able to modify the startup process of the bot so that the bot will re-cache the animation and behavior information for all the objects that the bot is responsible for.

All in all, by making all of these changes, the bot can now be shutdown and restarted on a regular basis without the need to make any type of modification to the database. Additionally, even after the bot gets shutdown and restarted, we do not lose any information regarding any of the created interactive objects. Thus, the users' experience will not be affected at all when the bot gets restarted.

9.3 Consistency Check Between Database and QA World

Even after all these changes were made to the Quest Atlantis bot, the created interactive objects in Quest Atlantis could still potentially get out of sync with that of the database. In Quest Atlantis, users have the ability to delete objects as well as change the location and orientation of objects without going through the platform. Because of this, objects can potentially be in a different location and orientation from what it should be based on the database. Similarly, an object can be deleted from Quest Atlantis but still exist in the database. From this we can see the potential consistency problem between Quest Atlantis and the database.

To solve this, we modified the bot to periodically perform a consistency check between Quest Atlantis and the database. Essentially, this check did the following:

- For each object entry in the Owned_Objects table:
 - o Check if the object exists in the respective Quest Atlantis world
 - o If it does not exist, then delete that entry from the table.
 - o If it does exist, then check if the location and orientation of the object matches that of the table.
 - o For any location or orientation fields that does not match, modify the entry so that it does match.

This periodic consistency check ensures that the created interactive objects in Quest Atlantis will be in sync with that of the database.

10 Admin Interface

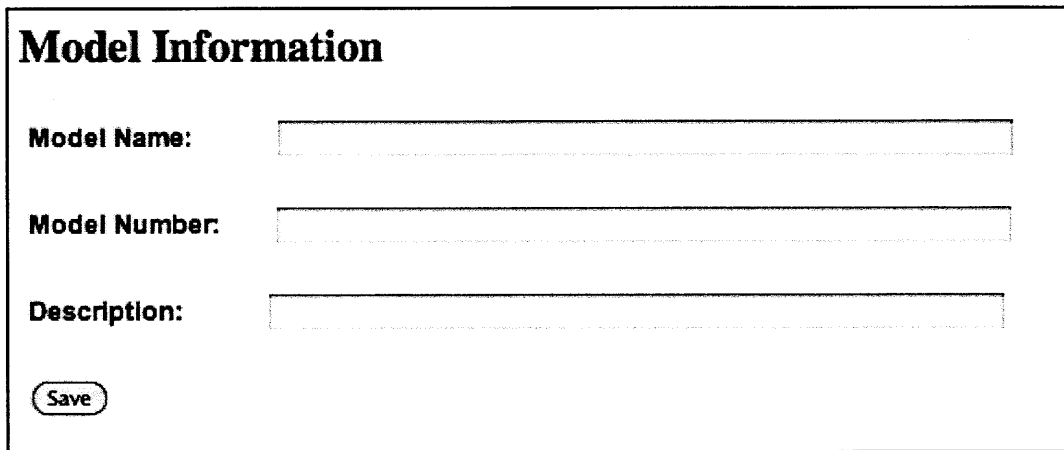
In building this intuitive fully integrated web platform that allows users to design and script their own interactive objects in Quest Atlantis, we also wanted to build an interface that allows an admin to make simple modifications to the platform. Specifically, an admin should have the ability to determine the types of objects that are available for users to create in Quest Atlantis and the animations and behaviors that are available for users to add into their interactive objects. In the existing platform, there is not an easy way for an admin to make these types of modifications. For example, if an admin wants to change what objects are available for users to create, the admin would have to modify the `Palette_Objects` table of the database. Similarly, if an admin wants to change what animations and behaviors for available for users to add into their objects, the admin would have to modify the XML file in the Scriptblocks system as well as modify the Quest Atlantis bot. Not only are these methods inconvenient and non-intuitive, but they can also be damaging to the system. By granting write permission to the database, the Scriptblocks XML file, and the Quest Atlantis bot, an admin has the potential to make accidental modifications that breaks the entire platform. As a result, we added an admin web interface to the platform that allows an admin to perform these simple modifications to the platform. This section details the design and development of this interface.

10.1 Creation of New Object Models

Before building the admin web platform, our original solution for easy modification of the types of objects that are available for users to create in Quest Atlantis was through the use of a Ruby YAML file. Since the `Palette_Objects` table of the database is what determines which objects are available to the user, we just put all the information that we wanted in the

Palette_Objects table of the database in a YAML file. Then, when needed, we used Ruby to read the information from this YAML file and re-populate the Palette_Objects table with the information. However, we quickly saw that this was not an ideal solution.

As a result, a core component of this new admin web interface is an interface that allows an admin to easily specify which objects are available for users to create. Shown in figure 15 is a screenshot of this interface.



The screenshot shows a web form titled "Model Information". It contains three input fields: "Model Name:", "Model Number:", and "Description:". Each field is followed by a horizontal text input box. At the bottom left of the form is a "Save" button.

Figure 15: Interface to Add New Object Types Into the Platform

Using this interface, an admin can now add a new object type in to the platform simply by entering the *name*, *model number*, and *description* of the object into the interface. The admin interface then reads in these values and modifies the Palette_Objects table of the database to reflect these new changes to the platform. Similarly, an admin is also able to modify or delete an existing object type through the home page of the admin web interface, which is discussed in section 10.4.

10.2 Creation of New Object Commands

As discussed earlier, the existing platform does not allow for easy modification of the animations and behaviors that are available for users to add into their objects. For example, if an admin wants to add a new animation/behavior to the platform, the admin would have to

modify the Scriptblocks XML file as well as modify the Quest Atlantis bot to include this new animation/behavior. However, giving an admin the ability to modify the Scriptblocks XML file and the Quest Atlantis bot can be potentially damaging to the system as a whole. In addition, it is unreasonable to expect an admin to have the computer science knowledge necessary to successfully modify the bot. Therefore, another core component of the admin web interface is an interface that allows an admin to easily specify the animations and behaviors that are available for users to add into their interactive objects.

As discussed in section 8.3, the Scriptblocks XML file contains the specifications of the command blocks that are available in the Scriptblocks system. Unfortunately, this type of setup did not work well for this new interface since designing this new interface to be able to modify the Scriptblocks XML file is no easy task. As a result, in building this new interface, we modified the start-up process of the Scriptblocks system to read in the list of available animations and behaviors from the database instead of the Scriptblocks XML file. Specifically, the Scriptblocks system will be reading from a newly created *Animations* table in the database.

The *Animations* table contains the specifications of all the command blocks that were originally placed in the Scriptblocks XML file. In other words, each entry in the *Animations* table contains information pertaining to a particular animation/behavior block in the Scriptblocks system. By having the new *animations* table dictate which command blocks are available to the Scriptblocks system, we are now able to easily add new commands as well as modify existing commands to the Scriptblocks system.

Unfortunately, even with this new *Animations* table in place, the modification and addition of commands that are available for users to add into their objects still involved the

modification of the Quest Atlantis bot. This is so because the programming logic for each command was hard-coded into the Quest Atlantis bot using a combination of Active Worlds Scripting Language and SDK. However, as we saw in section 5, most of the commands can be represented using Active Worlds Scripting Language. As a result, instead of hard-coding the corresponding Active Worlds Script for each command into the bot, we stored them into the database. This was accomplished by adding an additional *aw_script* field into the *Animations* table of the database.

id	animation	arg_name	arg_type	aw_script

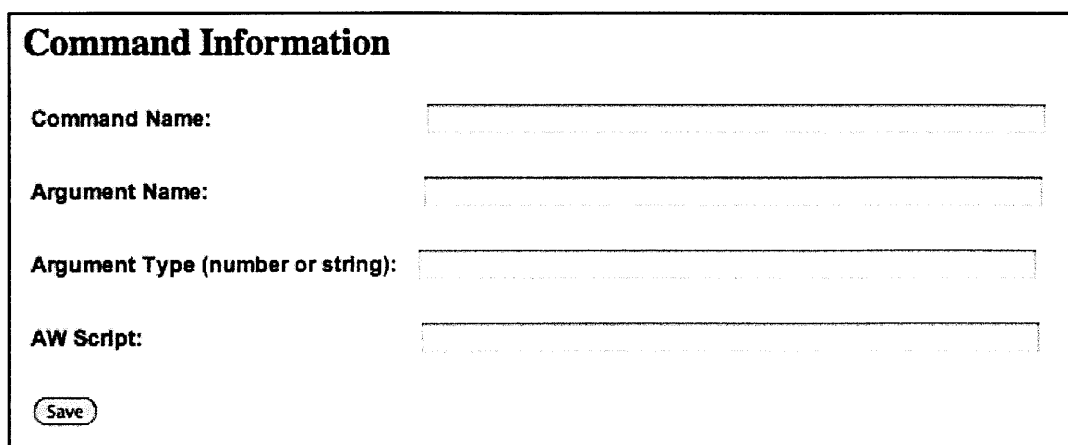
Figure 16: Schema of New Animations Table

While this new setup works for most commands, there still exists commands that cannot simply be represented strictly using Active Worlds Scripting Language. For example, the *transform* command, which is discussed in section 5.5, can only be implemented using Active Worlds SDK. As a result, commands like these were hard-coded into the Quest Atlantis bot and contained the record “None” or an empty record in the *aw_script* field of the corresponding entry in the *Animations* table.

After making these changes, we modified the Quest Atlantis bot to read in the *aw_script* for each command from the database. To do so, we modified the start-up process of the bot to read in each entry from the *Animations* table and store the (key, value) pair of (*animation*, [*arg_type*, *aw_script*]) in an in-memory hash table. By having all this information in memory, the Quest Atlantis bot can carry out each animation/behavior by taking the corresponding *aw_script* and executing it.

However, we did not want to have to restart the Quest Atlantis bot every time we add a new command or modify an existing command. In other words, any change we make to the *Animations* table of the database should be propagated to the Quest Atlantis bot in real-time. This was accomplished by using the *Q_Actions* table of the database. As discussed in section 3.3, the *Q_Actions* table is the primary method of communication between the different components of this platform and the Quest Atlantis bot. To communicate changes in the *Animations* table to the Quest Atlantis bot, a new type of request known as an *AW_SCRIPT_CHANGE* request was created. Each *AW_SCRIPT_CHANGE* request consisted of three entries with the keys *animation*, *arg_type*, and *script*. The values associated with these keys are the corresponding *animation*, *arg_type*, and *aw_script* fields of the *Animations* table. Once an *AW_SCRIPT_CHANGE* request is added into the *Q_Actions* table, the bot will process this request by adding these new values to the hash table.

By implementing all these changes to the different components of the platform, we created an interface that allows an admin to easily create new animations and behaviors for users to add into their interactive objects. Shown in figure 17 is a screenshot of this interface.



The screenshot shows a web form titled "Command Information" with a black border. It contains four input fields, each with a label to its left: "Command Name:", "Argument Name:", "Argument Type (number or string):", and "AW Script:". Each label is followed by a horizontal text input box. At the bottom left of the form is a "Save" button with a rounded rectangular border.

Figure 17: Interface to Add New Commands Into the Platform

Using this interface, an admin can now add a new animation/behavior in to the platform simply by entering the *name*, *argument name*, *argument type*, and *aw_script* of the command into the interface. The admin interface then reads in these values and modifies the *Animations* table of the database as well as propagates this new command to the Quest Atlantis bot. Similarly, an admin can now modify or delete an existing command through the home page of the admin web interface, which is discussed in section 10.4. However, as mentioned earlier, keep in mind that this interface only works for commands that can be represented using Active Worlds Scripting Language.

10.3 Ability to Select Which Commands Each Model Can Have

In addition to being able to determine the types of objects that are available for users to create in Quest Atlantis and the animations and behaviors that are available for users to add into their interactive objects, an admin should also be able to use this new admin interface to determine which commands a specific object type should have. The implementation changes to the platform discussed in sections 10.2 only allowed an admin to determine which animation/behavior were available for users to add into any interactive object created in Quest Atlantis. However, there are certain animation/behavior that an admin may wish to only be available for certain object types. For example, an admin may want the walk command to only be available to avatar object types. Therefore, as part of this new admin interface, for any given object type, an admin is able to select which commands are available for that object. Similarly, for any command that an admin creates using this new admin interface, the admin is able to select which object types are allowed to have this command.

To add this functionality to the admin interface, we first created a many-to-many mapping between the *Animations* table and the *Palette_Objects* table of the database.

Essentially, this allowed a given palette_object to be associated with many animations and similarly a given animation to be associated with many palette_objects. After creating this mapping in the database, we modified the Scriptblocks system so that when a user opens up the Scriptblocks interface for a given object, the Scriptblocks system will use this new mapping in the database to determine which command blocks it should display for that object.

Command Information

Command Name:

Argument Name:

Argument Type (number or string):

AW Script:

Models With this Command

Select which model has this command

- Bench
- Lamp
- Tree
- Car
- Elf
- Girl
- WaterLady

Figure 18: An admin is now able to select which object types can have a given command

Model Information

Model Name:

Model Number:

Description:

Available Commands

Select which commands this model can have

- SlideForward
- SlideBackward
- TurnLeft
- TurnRight
- Walk
- PlaySound
- Say
- Teleport_X
- Teleport_Y
- Transform
- Scale

Save

Figure 19: An admin is now able to select which commands a given object type can have

10.4 Admin Home

Finally, using this new admin web interface, an admin should be able to view or modify all of the existing types of objects that are available for users to create in Quest Atlantis and the existing animations and behaviors that are available for users to add into their interactive objects. The implementation changes to the platform discussed in section 10.1, 10.2, and 10.3 only allowed an admin to create and add new object types and commands into the platform. As a result, as part of this new admin web interface, we built a homepage that contains a listing of all the existing object types and commands in the platform. Additionally, next to each object type and command listing is an edit and delete link that allows an admin to modify or delete the corresponding object type or command. As we can see, this homepage is essentially what ties all the other components of the admin web interface together.

QA Models Listing						
Name	Model Number	Description	Available Commands		Edit	Delete
Bench	fzol_bench1.rwx	Bench	Teleport_X, Say, PlaySound, TurnRight, TurnLeft, Scale, Transform, Teleport_Y, SlideBackward, SlideForward		Edit	Delete
Lamp	mblamp4lag	Lamp	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound, Scale, Teleport_Y, Teleport_X		Edit	Delete
Tree	poak4	Tree	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound, Scale, Teleport_Y, Teleport_X		Edit	Delete
Car	cd_car0075	Car	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound		Edit	Delete
Elf	malelf1	Elf	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound, Say, Walk		Edit	Delete
Girl	mod_girl1	Girl	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound, Walk, Say		Edit	Delete
WaterLady	waterlady_01	WaterLady	SlideForward, SlideBackward, TurnLeft, TurnRight, PlaySound, Walk, Say		Edit	Delete

QA Commands Listing						
Command	Arg Name	Arg Type	Script	Models With This Command	Edit	Delete
SlideForward	steps	number	move 0 0 #[parameter] wait=2 ltm global	WaterLady, Girl, Elf, Car, Tree, Lamp, Bench	Edit	Delete
SlideBackward	steps	number	move 0 0 -#[parameter] wait=2 ltm global	WaterLady, Girl, Elf, Car, Tree, Lamp, Bench	Edit	Delete
TurnLeft	degs	number	rotate #[parameter/6] time=1 wait=2 global	WaterLady, Girl, Elf, Car, Tree, Lamp, Bench	Edit	Delete
TurnRight	degs	number	rotate -#[parameter/6] time=1 wait=2 global	WaterLady, Girl, Elf, Car, Tree, Lamp, Bench	Edit	Delete
Walk	steps	number	seq walk, move 0 0 #[parameter] time=1.5 wait=2 ltm global	WaterLady, Girl, Elf	Edit	Delete
PlaySound	arg	string	noise #[parameter] global	WaterLady, Girl, Elf, Car, Tree, Lamp, Bench	Edit	Delete
Say	arg	string	say `#[parameter]`	WaterLady, Girl, Elf, Bench	Edit	Delete
Teleport_X	steps	number	warp +#[parameter] +0 1.5a	Bench, Tree, Lamp	Edit	Delete
Teleport_Y	steps	number	warp +0 +#[parameter] 1.5a	Bench, Tree, Lamp	Edit	Delete
Transform			None	Bench	Edit	Delete
Scale	ratio	number	scale #[parameter]	Tree, Lamp, Bench	Edit	Delete

[Add new model](#) [Add new command](#)

Figure 20: Home Page of the New Admin Interface

11 Conclusion

The successful completion of this thesis project has added a whole new dimension into Quest Atlantis. The addition of this intuitive fully integrated platform for designing interactive objects in Quest Atlantis will now allow users to create and design their own objects in Quest Atlantis without the need to learn any programming language or complex platform. Users of the platform will be able to use their creativity in Quest Atlantis and they will also be introduced to fundamental computer programming concepts and skills. As a result, this will make Quest Atlantis an even more important teaching tool, specifically in the area of computer science.

Previous work has been done to build a simple platform that allows users to design and script their own interactive objects so that they are able to produce their own interactive narrative. Specifically, previous work has been done to build the Ruby-on-Rails web application, the Flashblocks system, and the Quest Atlantis bot. The Ruby-on-Rails web application allowed users to create and add interactive objects into Quest Atlantis. The Flashblocks system allowed users to add sequences of animations and behaviors into their interactive objects. The Quest Atlantis bot facilitated communication from the web application and the Flashblocks to Quest Atlantis as well as handled event triggers and object animations and behaviors.

However, this previous platform was immature and lacked many important needed features. As a result, using this previous platform as the groundwork, my role was to work with the designers of Quest Atlantis to build a fully-integrated platform that is not only easy to use, but also contains all the features a user would need. In summary, the following are the main additions I made to the platform.

- Creation of new animations and behaviors that users are able to add into their interactive objects
- Creation of a web interface that allows users to modify any object's basic property, animation/behavior, and position
- Creation of user permissions to limit users to only be allowed to modify the interactive objects that they created and ones that their teammates created
- Replacement of the Flashblocks system with the Scriptblocks system since the Scriptblocks system contained the needed flexibility and functionality that the Flashblocks system lacked
- Re-engineering of the Quest Atlantis bot so that restarting the bot will have no impact on the platform
- Creation of an admin interface that allows an admin to determine the types of interactive objects that are available for users to create in Quest Atlantis and the animations and behaviors that are available for users to add into their interactive objects

These additions to the platform have turned the platform into a powerful fully integrated tool that allows users to create and design their own objects in Quest Atlantis. However, with that said, there is still much work that can be done to improve the platform as a whole. One improvement is the user interface of the Scriptblocks system. Work can be done to make the user interface of the Scriptblocks system much more intuitive. Another improvement is the admin interface. Work can be done to make the admin interface a much more powerful tool for an admin. For example, a data-mining component can be added to the system that gives meaningful data regarding how users are using the platform.

12 References

[1] Fisher, Timothy. Ruby on Rails Bible. Indianapolis, IN: Wiley Publishing, 2008.

[2] Rails API Documentation. <<http://api.rubyonrails.org/>>

[3] Active Worlds Documentation.

<http://wiki.activeworlds.com/index.php?title=Main_Page>

[4] Active Worlds Object Scripting.

<http://wiki.activeworlds.com/index.php?title=Object_Scripting>

[5] Active Worlds SDK.

<<http://wiki.activeworlds.com/index.php?title=SDK>>

[6] Active Worlds Say Command.

<<http://wiki.activeworlds.com/index.php?title=Say>>

[7] Active Worlds Noise Command.

<<http://wiki.activeworlds.com/index.php?title=Noise>>

[8] Active Worlds Seq Command.

<[http://wiki.activeworlds.com/index.php?title=Seq_\(building_command\)](http://wiki.activeworlds.com/index.php?title=Seq_(building_command))>

[9] Active Worlds Move Command.

<<http://wiki.activeworlds.com/index.php?title=Move>>

[10] Active Worlds Warp Command.

<<http://wiki.activeworlds.com/index.php?title=Warp>>

[11] Active Worlds Scale Command.

<<http://wiki.activeworlds.com/index.php?title=Scale>>

[12] Active Worlds Media Command.

<<http://wiki.activeworlds.com/index.php?title=Media>>

[12] Javascript Numeric Stepper.

<<http://www.htmldrive.net/items/show/540/Javascript-numeric-stepper-with-inputbox.html>>

[13] Irizarry, Angel, “Intuitive Interface for Object Interactivity and Storytelling for Quest Atlantis”, Master’s Thesis, Massachusetts Institute of Technology, 2010

[14] Roque, Ricarose Vallarata, “OpenBlocks : an extendable framework for graphical block programming systems”, Master’s Thesis, Massachusetts Institute of Technology, 2007

[15] Barab, Sasha, “Transactive Narrative Art Proposal”, Indiana University

A Appendix – Setup

As discussed in section 4.3, the Ruby-on-Rails web application is currently being hosted on `scripts.mit.edu`. Specifically, the web application that allows users to create and edit their own interactive objects is hosted on `http://d_lam201.scripts.mit.edu/qaeditor`. Additionally, the admin web application that allows admin to make simple modifications to the platform is hosted on `http://d_lam201.scripts.mit.edu/qaeditor/admin`. The database for these web applications is currently being hosted on `sql.mit.edu`.

To setup the necessary environment to run the Quest Atlantis bot, please follow the instructions listed in section 4.1. The source code for the Quest Atlantis bot is no longer hosted on github and is instead hosted on the `education.mit.edu/svn/newqa` svn server. Similarly, the source code for the Ruby-on-Rails web application is also hosted on the same svn server. To get access to this svn server, one must create a username/password combination. One can do this by going to the following website to generate a username/password hash combo: `http://www.4webhelp.net/us/password.php`. Once the username/password hash combo has been generated, it must be sent to Professor Klopfer (`klopfer@mit.edu`).

Having the source code, one can setup the Ruby-on-Rails web application as well as the database on a different server. However, in doing so, one must make modify the database configuration file to point to the new database.

B Source Code – Quest Atlantis Bot

B.1 bot_db.rb

```
require 'rubygems'
require 'active_record'

#TODO: Grab these values from the rails YAML file
ActiveRecord::Base.establish_connection(
  :adapter => "mysql",
  :host    => "sql.mit.edu",
  :database => "d_lam201+qaeditor_development",
  :username => "d_lam201",
  :password => "password"
)

class QAction < ActiveRecord::Base

  # Grabs the first message off of the queue and all other messages
  # with the same action id. They form a set, where each message holds
  # a key/value pair that acts as a parameter for the message set.
  # See: process_message_set(msg_set)
  def self.new_message_set
    if QAction.exists?
      return QAction.find(:all, :conditions => { :action_id => QAction.first.action_id})
    else
      return [];
    end
  end

end

class PaletteObject < ActiveRecord::Base
end

class OwnedObject < ActiveRecord::Base

  def self.get_all_objects
    if OwnedObject.exists?
      return OwnedObject.all
    else
      return [];
    end
  end

end

class Animation < ActiveRecord::Base

  def self.get_all_commands
    if Animation.exists?
      return Animation.all
    else
      return [];
    end
  end

end

class ObjectPermission < ActiveRecord::Base

end
```

B.2 object_editor.rb

```
#!/usr/bin/env ruby1.9.1
require 'rubygems'
require 'activeworlds_ffi'
require 'bot_db'

include ActiveworldsFFI

$log = Logger.new(STDOUT)
$log.level = Logger::DEBUG

##### Globals #####

# This stores the AW scripts for each command
$commands = Hash.new()
$commands.default = {}

# This stores a hash of the information programmed (using blocks)
# for each owned object.
$object_actions = Hash.new()
$object_actions.default = {}

# The amount of time animations wait at their end before the bot permanently
# changes the objects' states. (in seconds)
$WAIT = 2

# The amount of time the bot waits between each check to the database.
# (in milliseconds)
$AW_WAIT = 200

##### TRIGGER HANDLERS #####

def handle_activate
  handle_trigger("WhenClicked")
end

def handle_right_away
  handle_trigger("RightAway")
end

def handle_bump
  handle_trigger("WhenBumped")
end

def handle_trigger(trigger)

  o = OwnedObject.find(:first, \
                      :conditions => \
                      [ "aw_object_id = ?", aw_int( AW_OBJECT_ID ) ])

  if !(o.nil?)
    $log.debug("Right Away handler called for object #{o.aw_object_id}.")
  end

  # Save the script of the current object so it can be restored to its original
  # state after all of the animations for this trigger are over. To make sure
  # the create-triggered actions only happen once, those scripts are removed.
  first_script = aw_string( AW_OBJECT_ACTION )
  first_script.gsub(/create.*; activate/, "activate")

  if !(o.nil?)
    actions_hash = $object_actions[o.aw_object_id]
    if (actions_hash.keys.length > 0)
```

```

actions_array = actions_hash[trigger];
unless (actions_array.nil?)
  Thread.new(o, actions_array) { |my_object, my_actions_array|
    first_action = true
    my_actions_array.each do |action_combo|

      if (first_action)
        $log.debug("First action. Skipping animation setup.")
        first_action = false
      else
        # Set up the next animation.
        $log.debug("Next action. Setting up the next animation #{action_combo[0]} with param
#{action_combo[1]} for #{o.aw_object_id}")
        script = script_string("RightAway", action_combo[0], action_combo[1]) + "; "

        # Activate the animation
        aw_int_set( AW_OBJECT_ID, my_object.aw_object_id )
        aw_int_set( AW_OBJECT_NUMBER, 0 )
        rc = aw_object_query()
        if (rc != RC_SUCCESS)
          $log.error("Tried to animate an object that no longer exists. (reason #{rc})")
        else
          aw_string_set( AW_OBJECT_ACTION, script )
          rc = aw_object_change()
          if (rc != RC_SUCCESS)
            $log.error("Something went wrong during an animation transition point. (reason #{rc})")
          else
            $log.debug("Object #{my_object.aw_object_id} set to next animation: <#{script}>")
          end
        end
      end
    end
  }
end

# Wait until the animation is over, then make the permanent change.
sleep(1)
apply_action(my_object, action_combo)
end

# When all animations are over, reset the object to contain the script
# it started with. This allows the object to be retriggered by the same
# trigger.
aw_int_set( AW_OBJECT_ID, my_object.aw_object_id )
aw_int_set( AW_OBJECT_NUMBER, 0 )
rc = aw_object_query()
if (rc != RC_SUCCESS)
  printf("Tried to animate an object that no longer exists.\n", rc);
else
  aw_string_set( AW_OBJECT_ACTION, first_script )
  rc = aw_object_change()
  if (rc != RC_SUCCESS)
    printf("Something went wrong during an animation transition point.\n")
  else
    puts "Transition triggered!"
  end
end
end
}
end
end
end
end
end

```

```
##### Utility functions #####
```

```

def script_string(trigger, action, parameter)
  script = ""
  trigger = case trigger
    when "WhenClicked"
      "activate"
    when "WhenBumped"
      "bump"
    when "RightAway"

```

```

        "create"
    end

    if ($commands.has_key?(action))
        command_info = $commands[action]
        command = ""

        if(command_info["Arg_Type"] == "number")
            l_index = command_info["Script"].index("#[")
            r_index = command_info["Script"].index("]")

            input = command_info["Script"][l_index, r_index - l_index + 1]
            input = input[2, input.length - 3]
            input = input.sub("parameter", parameter.to_s)
            input = eval(input)

            command = command_info["Script"][0, l_index] + input.to_s + command_info["Script"][r_index + 1,
            command_info["Script"].length - r_index - 1]

        else
            command = command_info["Script"]
            command = command.sub("#[parameter]", parameter.to_s)
        end

        script = "#{trigger} #{command}"

    else

        #AW Script Doesn't Exist Cuz the Command is too Complex

    end

    return script
end

def create_object(obj_name, obj_x, obj_z)
    aw_int_set( AW_OBJECT_X, obj_x * 100 )
    aw_int_set( AW_OBJECT_Y, 0 )
    aw_int_set( AW_OBJECT_Z, obj_z * 100 )
    aw_int_set( AW_OBJECT_YAW, 0 )
    aw_int_set( AW_OBJECT_TILT, 0 )
    aw_int_set( AW_OBJECT_ROLL, 0 )
    aw_string_set(AW_OBJECT_MODEL, obj_name )
    aw_string_set( AW_OBJECT_DESCRIPTION, "" )
    rc = aw_object_add()
    if (rc != RC_SUCCESS)
        printf("Unable to add object (reason %d)\n", rc)
        o = OwnedObject.find(:first, \
            :conditions => \
            [ "x = ? AND z = ?", obj_x, obj_z ])

        o.delete
    else
        obj_id = aw_int( AW_OBJECT_ID )
        $log.debug("Object #{obj_id} added to location #{obj_x}, #{obj_z}.")
        o = OwnedObject.find(:first, \
            :conditions => \
            [ "x = ? AND z = ?", obj_x, obj_z ])

        o.aw_object_id = obj_id
        o.save

        # Allow editing
        aw_int_set( AW_OBJECT_NUMBER, 0 )
        aw_int_set( AW_OBJECT_OLD_X, 0 )
        aw_int_set( AW_OBJECT_OLD_Z, 0 )

        # default_action = "activate " + default_action_string(obj_id) + ";"
        # aw_string_set( AW_OBJECT_ACTION, default_action )
        rc = aw_object_change()
        if (rc != RC_SUCCESS)

```

```

        $log.error("Unable to add the default action to object (reason #{rc}).")
    end
end
end

def delete_object(o)
    if ( o )
        aw_int_set( AW_OBJECT_ID, o.aw_object_id )
        aw_int_set( AW_OBJECT_NUMBER, 0 )
        rc = aw_object_delete()
        if (rc != RC_SUCCESS)
            $log.error("Unable to delete object (reason #{rc}).")

        else
            $log.debug("Object #{o.aw_object_id} deleted.")
        end
        o.delete
    end
end

def change_object(o, changes)
    if ( o )

        aw_int_set( AW_OBJECT_ID, o.aw_object_id )
        aw_int_set( AW_OBJECT_NUMBER, 0 )
        aw_int_set( AW_OBJECT_TYPE, AW_OBJECT_TYPE_V3 )

        aw_string_set( AW_OBJECT_MODEL, changes[:model].to_s ) unless (changes[:model].nil?)
        aw_string_set( AW_OBJECT_DESCRIPTION, changes[:description].to_s ) unless (changes[:description].nil?)

        aw_int_set( AW_OBJECT_X, changes[:x].to_i * 100 ) unless (changes[:x].nil?)
        aw_int_set( AW_OBJECT_Y, changes[:y].to_i * 100 ) unless (changes[:y].nil?)
        aw_int_set( AW_OBJECT_Z, changes[:z].to_i * 100 ) unless (changes[:z].nil?)

        aw_int_set( AW_OBJECT_YAW, changes[:yaw].to_i * 10 ) unless (changes[:yaw].nil?)
        aw_int_set( AW_OBJECT_TILT, changes[:tilt].to_i * 10 ) unless (changes[:tilt].nil?)
        aw_int_set( AW_OBJECT_ROLL, changes[:roll].to_i * 10 ) unless (changes[:roll].nil?)

        # Unset the action field of the object
        aw_string_set(AW_OBJECT_ACTION, "")

        rc = aw_object_change()
        if (rc != RC_SUCCESS)
            $log.error("Unable to update object (reason #{rc}).")
        else
            $log.debug("Object #{o.aw_object_id} updated!.");
        end

        if(!changes[:action].nil?)
            $log.debug("Calling script_object on #{o.aw_object_id} with script: <#{changes[:action].to_s}>.")
            script_object(o, changes[:action].to_s)
        end
    end
end

def cache_object_animations(o, action_string)
    actions = {}
    blocks_array = action_string.scan(/\{.+?\}/)
    blocks_array.each do |block_string|
        block_string.scan(/\{(.)\[(.+)\]\}/)
        trigger = $1
        commands = $2.split(",")
        commands.each do |command_combo|
            command_combo.scan(/(.+)\((.+)\)/)
            actions[trigger] ||= []
            actions[trigger] << [$1, $2]
        end
    end
end

```

```

# Save the actions globally
$object_actions[o.aw_object_id] = actions
end

def script_object(o, action_string)
  actions = {}
  blocks_array = action_string.scan(/\{.+?\}/)
  blocks_array.each do |block_string|
    block_string.scan(/\{(.+)\[(.+)\]\}/)
    trigger = $1
    commands = $2.split(",")
    commands.each do |command_combo|
      command_combo.scan(/(.+)\((.+)\)/)
      actions[trigger] ||= []
      actions[trigger] << [$1, $2]
    end
  end
end

# Save the actions globally
$object_actions[o.aw_object_id] = actions

# In this section we check each trigger in order and build
#the script to insert into the object. Order matters.
script = ""

# The "Right Away" Trigger
trigger = "RightAway"
actions_array = actions[trigger]
unless actions_array.nil?
  action = actions_array[0][0]
  parameter = actions_array[0][1]
  script += script_string(trigger, action, parameter) + ";"
end

# The "When Clicked" Trigger
trigger = "WhenClicked"
actions_array = actions[trigger]
unless actions_array.nil?
  action = actions_array[0][0]
  parameter = actions_array[0][1]
  script += script_string(trigger, action, parameter) + ";"
end

# The "When Bumped" Trigger
trigger = "WhenBumped"
actions_array = actions[trigger]
unless actions_array.nil?
  action = actions_array[0][0]
  parameter = actions_array[0][1]
  script += script_string(trigger, action, parameter) + ";"
end

$log.debug("Object #{o.aw_object_id} is about to get scripted with: <#{script}>.")

# Inject the script into the object
aw_int_set( AW_OBJECT_ID, o.aw_object_id )
aw_int_set( AW_OBJECT_NUMBER, 0 )
aw_int_set( AW_OBJECT_TYPE, AW_OBJECT_TYPE_V3 )

aw_string_set( AW_OBJECT_ACTION, script )
rc = aw_object_change()
if (rc != RC_SUCCESS)
  $log.error("Unable to add script the object (reason #{rc}).")
else
  $log.debug("Object #{o.aw_object_id} successfully scripted!")
  o.actions = script
  o.save
end
if actions["RightAway"]

```



```

    handle_right_away()
end

end

def add_aw_script(cmd_name, arg_type, aw_script)
  command_info = {}
  command_info["Arg_Type"] = arg_type
  command_info["Script"] = aw_script
  $log.debug("For command: <#{cmd_name}>, storing aw script: <#{aw_script}>")
  $commands[cmd_name] = command_info
end

def apply_action(o, action_combo)

  action = action_combo[0]
  parameter = action_combo[1]
  changes = {}

  aw_script = $commands[action]["Script"]

  # Check to see if the aw script contains a move command
  if(!aw_script.index("move").nil?)

    # Check to see if it is suppose to move forward or backward
    if(aw_script[aw_script.index("#[") - 1, 1] == "-")

      # Move backward change
      parameter = parameter.to_i
      angle = (Math::PI * ((o.yaw + 180) % 360)/180)
      o.x += parameter * Math.sin(angle)
      o.z += parameter * Math.cos(angle)
      changes[:x] = o.x
      changes[:z] = o.z
    else

      # Move forward change
      parameter = parameter.to_i
      angle = Math::PI * o.yaw/180
      o.x += parameter * Math.sin(angle)
      o.z += parameter * Math.cos(angle)
      changes[:x] = o.x
      changes[:z] = o.z
    end

    change_object(o, changes)
    o.save

  #Check to see if the aw script contains a rotate command
  elsif(!aw_script.index("rotate").nil?)

    # Check to see if it is rotate left or rotate right
    if(aw_script[aw_script.index("#[") - 1, 1] == "-")

      # Rotate right change
      parameter = parameter.to_i
      o.yaw += (360 - parameter)
      o.yaw = o.yaw % 360
      changes[:yaw] = o.yaw
    else

      # Rotate left change
      parameter = parameter.to_i
      o.yaw += parameter
      changes[:yaw] = o.yaw
    end

    change_object(o, changes)
    o.save
  end
end

```

```

end

end

##### Handlers #####

def consistency_check(o)
  aw_int_set( AW_OBJECT_ID, o.aw_object_id )
  aw_int_set( AW_OBJECT_NUMBER, 0 )
  rc = aw_object_query()
  if (rc != RC_SUCCESS)
    $log.debug("Object #{o.aw_object_id} no longer exists. Deleting it from the database")
    o.delete
  else
    if(o.x * 100 != aw_int(AW_OBJECT_X))
      $log.debug("Fixing inconsistent x location for object #{o.aw_object_id}")
      o.x = aw_int(AW_OBJECT_X)/100
      o.save
    end

    if(o.y * 100 != aw_int(AW_OBJECT_Y))
      $log.debug("Fixing inconsistent y location for object #{o.aw_object_id}")
      o.y = aw_int(AW_OBJECT_Y)/100
      o.save
    end

    if(o.z * 100 != aw_int(AW_OBJECT_Z))
      $log.debug("Fixing inconsistent z location for object #{o.aw_object_id}")
      o.z = aw_int(AW_OBJECT_Z)/100
      o.save
    end

    if(o.yaw * 10 != aw_int(AW_OBJECT_YAW))
      $log.debug("Fixing inconsistent yaw rotation for object #{o.aw_object_id}")
      o.yaw = aw_int(AW_OBJECT_YAW)/10
      o.save
    end

    if(o.tilt * 10 != aw_int(AW_OBJECT_TILT))
      $log.debug("Fixing inconsistent tilt rotation for object #{o.aw_object_id}")
      o.tilt = aw_int(AW_OBJECT_TILT)/10
      o.save
    end

    if(o.roll * 10 != aw_int(AW_OBJECT_ROLL))
      $log.debug("Fixing inconsistent roll rotation for object #{o.aw_object_id}")
      o.roll = aw_int(AW_OBJECT_ROLL)/10
      o.save
    end
  end
end

def process_message_set(msg_set)
  type = msg_set[0].name

  # Parse the message and make a hash of key/value pairs
  msg_hash = {}
  msg_set.each do |msg|
    msg_hash[msg.key.to_sym] = msg.value
  end

  # If a message matches one of the expected types, check to see
  # if the required paramaters exist, and if so, call the appropriate
  # method
  case type
  when "CREATE"
    # CREATE: x, z, name

```

```

$log.debug("Processing a CREATE message set.")
if (msg_hash[:x] and msg_hash[:z] and msg_hash[:name])
  name = msg_hash[:name]
  x = msg_hash[:x].to_i
  z = msg_hash[:z].to_i
  $log.debug("Calling create_object(#{name}, #{x}, #{z}).")
  create_object(name, x, z)
else
  return false
end

when "DELETE"
# DELETE: x, z
$log.debug("Processing a DELETE message set.")
if (msg_hash[:x] and msg_hash[:z])
  x = msg_hash[:x].to_i
  z = msg_hash[:z].to_i
  o = OwnedObject.find(:first, \
                        :conditions => \
                        [ "x = ? AND z = ?", x, z ])
  if (o)
    if(o.project_id == msg_hash[:project_id].to_i)
      $log.debug("Calling delete_object(#{o.aw_object_id})")
      delete_object(o)
    else
      $log.warn("Don't have permission to modify object")
      return true
    end
  else
    $log.warn("Attempted to delete an object at #{x}, #{z} that did not exist.")
    return true
  end
else
  return false
end

when "CHANGE"
# CHANGE: aw_object_id, description, action, x, y, z, yaw, tilt, roll
$log.debug("Processing a CHANGE message set.")
if (msg_hash[:aw_object_id] and msg_hash[:model] and msg_hash[:description] and msg_hash[:action] \
    and msg_hash[:x] and msg_hash[:y] and msg_hash[:z] \
    and msg_hash[:yaw] and msg_hash[:tilt] and msg_hash[:roll])
  obj_id = msg_hash[:aw_object_id]
  model = msg_hash[:model]
  description = msg_hash[:description]
  action = msg_hash[:action]
  x = msg_hash[:x]
  y = msg_hash[:y]
  z = msg_hash[:z]
  yaw = msg_hash[:yaw]
  tilt = msg_hash[:tilt]
  roll = msg_hash[:roll]
  o = OwnedObject.find(:first, \
                        :conditions => \
                        [ "aw_object_id = ?", obj_id ])

  if(o and o.project_id == msg_hash[:project_id].to_i)
    changes = {}
    changes[:model] = model
    changes[:description] = description
    changes[:action] = action
    changes[:x] = x
    changes[:y] = y
    changes[:z] = z
    changes[:yaw] = yaw
    changes[:tilt] = tilt
    changes[:roll] = roll
    $log.debug("Calling change_object on #{obj_id}.")
    change_object(o, changes)
  end
end

```

```

else
  $log.warn("Don't have permission to modify object")
  return true
end
else
  return false
end

when "SCRIPT"
# SCRIPT: aw_object_id, action_string
$log.debug("Processing a SCRIPT message set.")
if (msg_hash[:aw_object_id] and msg_hash[:action_string])
  obj_id = msg_hash[:aw_object_id].to_i
  action_string = msg_hash[:action_string]
  o = OwnedObject.find(:first, \
    :conditions => \
      [ "aw_object_id = ?", obj_id ])
  if(o and o.project_id == msg_hash[:project_id].to_i)
    $log.debug("Calling script_object on #{obj_id} with script: <#{action_string}>.")
    script_object(o, action_string)
  else
    $log.warn("Don't have permission to modify object")
    return true
  end
end
else
  return false
end

when "AW_SCRIPT_CHANGE"
# AW_SCRIPT_CHANGE: animation, arg_type, script
$log.debug("Processing a AW_SCRIPT_CHANGE message set.")
if (msg_hash[:animation] and msg_hash[:arg_type] and msg_hash[:script] and msg_hash[:script] != "" and
msg_hash[:script] != "None")
  cmd_name = msg_hash[:animation]
  arg_type = msg_hash[:arg_type]
  aw_script = msg_hash[:script]
  add_aw_script(cmd_name, arg_type, aw_script)
else
  return true
end
end
return true
end

##### Login & Setup #####

if ARGV.nil? || ARGV.size < 3
  puts("Usage: hello.rb citizen_id privilege_password world")
  exit(1)
end

# initialize Active Worlds API
rc = aw_init(AW_BUILD)
if( rc != RC_SUCCESS )
  printf("Unable to initialize API (reason %d)\n", rc)
  exit( 1 )
end

# assign the proc to a constant so that it never gets garbage collected
ACTIVATE_HANDLER = Proc.new { handle_activate }
BUMP_HANDLER = Proc.new { handle_bump }

# install handler for avatar_add event
aw_event_set( AW_EVENT_OBJECT_BUMP, BUMP_HANDLER )
aw_event_set( AW_EVENT_OBJECT_CLICK, ACTIVATE_HANDLER )

# create bot instance
prc = aw_create("atlantis.activeworlds.com", 5870, nil);
if rc != RC_SUCCESS

```

```

    printf "Unable to create bot instance (reason %d)\n", rc
    exit 1
end

# log bot into the universe
aw_int_set AW_LOGIN_OWNER, ARGV[0].to_i
aw_string_set AW_LOGIN_PRIVILEGE_PASSWORD, ARGV[1]
aw_string_set AW_LOGIN_APPLICATION, "Object Editor Bot"
aw_string_set AW_LOGIN_NAME, "ObjectEditorBot"

#just added this line to see if global vs. non-global state was the issue
aw_bool_set AW_ENTER_GLOBAL, 1

rc = aw_login
if rc != RC_SUCCESS
    printf("Unable to login (reason %d)\n", rc)
    exit(1)
end

# log bot into the world named on the command line
rc = aw_enter(ARGV[2]);
if (rc != RC_SUCCESS)
    printf("Unable to enter world (reason %d)\n", rc)
    exit(1)
end

# announce our position in the world
aw_int_set(AW_MY_X, 1000) /* 1W */
aw_int_set(AW_MY_Z, 1000) /* 1N */
aw_int_set(AW_MY_YAW, 2250) /* face towards GZ */
rc = aw_state_change
if rc != RC_SUCCESS
    printf "Unable to change state (reason %d)\n", rc
    exit 1
end

# Allow 3-axis rotation globally
aw_int_set( AW_WORLD_ALLOW_3_AXIS_ROTATION, 1 )

# Check for object consistency between the database and the world
object_set = OwnedObject.get_all_objects
object_set.each do |object|
    consistency_check(object)
end

# Cached the animations for all objects into memory
object_set = OwnedObject.get_all_objects
object_set.each do |object|
    if(!object.blockString.nil?)
        $log.debug("Caching animation: <#{object.blockString}> for object: #{object.aw_object_id}")
        cache_object_animations(object, object.blockString)
    end
end

# Read in the aw_script for each command and store it in memory
command_set = Animation.get_all_commands
command_set.each do |cmd|
    if(cmd.script.nil? or cmd.script == "" or cmd.script == "None")
        $log.debug("No aw script for command: <#{cmd.animation}>")
    else
        add_aw_script(cmd.animation, cmd.arg_type, cmd.script)
    end
end

$log.info("Logged in and ready.")

# /* main event loop */
begin

```

```

counter = 0
while aw_wait($AW_WAIT) == RC_SUCCESS
  # Process new actions
  msg_set = QAction.new_message_set
  unless msg_set.empty?
    success = process_message_set(msg_set)

    if success
      msg_set.each { |msg| msg.delete }
      $log.debug("Successfully processed a message set")
    end
  end

  # Perform a periodic consistency check
  if(counter == 100)
    object_set = OwnedObject.get_all_objects
    object_set.each do |object|
      consistency_check(object)
    end
    counter = 0
  else
    counter += 1
  end
end
ensure

aw_destroy
aw_term
end

```

C Source Code – Web-App Ruby Controllers

C.1 admin_controller.rb

```
class AdminController < ApplicationController

  def listing
    @models = PaletteObject.all
    @animations = Animation.all
  end

  def model
    @commands = Animation.all
    @model
    @model_commands
    if(!params[:palette_model_name].nil?)
      @model = PaletteObject.find(:first, \
        :conditions => \
        ["model_name = ?", params[:palette_model_name]])
      @model_commands = @model.animations
    end

    if request.post?
      @model = PaletteObject.find(:first, \
        :conditions => \
        ["model_name = ?", params[:model][:model_name]])

      if @model.nil?
        p = PaletteObject.new
        p.pretty_name = (params[:model][:pretty_name]).to_s
        p.model_name = (params[:model][:model_name]).to_s
        p.description = (params[:model][:description]).to_s
        p.save

        @commands.each do |c|
          if(params[c.animation])
            p.animations.push(c)
            p.save
          end
        end
      else
        @model.pretty_name = (params[:model][:pretty_name]).to_s
        @model.model_name = (params[:model][:model_name]).to_s
        @model.description = (params[:model][:description]).to_s
        @model.save

        @model.animations.clear

        @commands.each do |c|
          if(params[c.animation])
            @model.animations.push(c)
            @model.save
          end
        end
      end

      redirect_to :controller => 'admin', :action => 'listing'
    end
  end

  def command
    @models = PaletteObject.all
  end
end
```

```

@command
@command_models
if(!params[:animation].nil?)
  @command = Animation.find(:first, \
                           :conditions => \
                           [ "animation = ?", params[:animation]])
  @command_models = @command.palette_objects
end

if request.post?
  @command = Animation.find(:first, \
                           :conditions => \
                           [ "animation = ?", params[:command][:animation]])

  if @command.nil?
    c = Animation.new
    c.animation = (params[:command][:animation]).to_s
    c.arg_name = (params[:command][:arg_name]).to_s
    c.arg_type = (params[:command][:arg_type]).to_s
    c.script = (params[:command][:script]).to_s
    c.save

    @models.each do |m|
      if(params[m.pretty_name])
        c.palette_objects.push(m)
        c.save
      end
    end

  else
    @command.animation = (params[:command][:animation]).to_s
    @command.arg_name = (params[:command][:arg_name]).to_s
    @command.arg_type = (params[:command][:arg_type]).to_s
    @command.script = (params[:command][:script]).to_s
    @command.save

    @command.palette_objects.clear

    @models.each do |m|
      if(params[m.pretty_name])
        @command.palette_objects.push(m)
        @command.save
      end
    end
  end

  # Push this aw_script change into the Q_Actions table so
  # that the bot can act on it

  action_id = rand(10000000)

  # animation name
  action = QAction.new do |a|
    a.name = 'AW_SCRIPT_CHANGE'
    a.action_id = action_id
    a.key = "animation"
    a.value = (params[:command][:animation]).to_s
  end
  action.save

  # arg type
  action = QAction.new do |a|
    a.name = 'AW_SCRIPT_CHANGE'
    a.action_id = action_id
    a.key = "arg_type"
    a.value = (params[:command][:arg_type]).to_s
  end
end

```



```

    action.save

    # aw script
    action = QAction.new do |a|
      a.name = 'AW_SCRIPT_CHANGE'
      a.action_id = action_id
      a.key = "script"
      a.value = (params[:command][:script]).to_s
    end
    action.save

    redirect_to :controller => 'admin', :action => 'listing'

  end
end

def delete_model
  @model = PaletteObject.find(:first, \
                              :conditions => \
                              [ "model_name = ?", params[:palette_model_name]])

  @model.animations.clear
  @model.delete

  redirect_to :controller => 'admin', :action => 'listing'
end

def delete_command
  @command = Animation.find(:first, \
                            :conditions => \
                            [ "animation = ?", params[:animation]])

  @command.palette_objects.clear
  @command.delete

  redirect_to :controller => 'admin', :action => 'listing'
end
end

```

C.2 objects_controller.rb

```

# require 'rubygems'

class ObjectsController < ApplicationController

  #Main screen that contains the editor grid and the palette of PaletteObjects
  def edit
    @palette_objects = PaletteObject.all
  end

  #Screen listing all the objects a user has created
  def listing
    @created_objects = Project.find(:first, \
                                    :conditions => \
                                    [ "name = ?", "Test_Project"]).owned_objects
  end

  # Edit all of the parameters of an OwnedObject. This screen is opened when clicked by an object.
  def tweak
    @qa_object = OwnedObject.find(:first, \
                                   :conditions => \
                                   [ "aw_object_id = ?", params[:aw_obj_id] ])

    if request.post?
      puts "tweak post request"
      action_id = rand(10000000)

      # aw_object_id

```

```

action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "aw_object_id"
  a.value = (params[:qa_object][:aw_object_id].to_i).to_s
end
action.save

# name
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "name"
  a.value = (params[:qa_object][:name]).to_s
end
action.save

# model
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "model"
  a.value = (params[:qa_object][:model]).to_s
end
action.save

# description
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "description"
  a.value = (params[:qa_object][:descriptions]).to_s
end
action.save

# action
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "action"
  a.value = (params[:qa_object][:blockString]).to_s
end
action.save

# x location
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "x"
  a.value = (params[:qa_object][:x].to_i).to_s
end
action.save

# y location
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "y"
  a.value = (params[:qa_object][:y].to_i).to_s
end
action.save

# z location
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "z"
  a.value = (params[:qa_object][:z].to_i).to_s
end

```

```

action.save

# yaw
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "yaw"
  a.value = (params[:qa_object][:yaw].to_i).to_s
end
action.save

# tilt
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "tilt"
  a.value = (params[:qa_object][:tilt].to_i).to_s
end
action.save

# roll
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "roll"
  a.value = (params[:qa_object][:roll].to_i).to_s
end
action.save

# project_id
action = QAction.new do |a|
  a.name = 'CHANGE'
  a.action_id = action_id
  a.key = "project_id"
  a.value = 1
end
action.save

@qa_object.update_attributes(params[:qa_object])
end
end

def saveScript
  #Save the xml string of the block structure
  @qa_object = OwnedObject.find(:first, \
    :conditions => \
    [ "aw_object_id = ?", params[:aw_obj_id] ])
  @qa_object.blockXML = params[:blockXML]
  @qa_object.blockString = params[:blockString]
  @qa_object.save

  # Add script action into the queue
  action_id = rand(10000000)

  # aw_object_id
  action = QAction.new do |a|
    a.name = 'SCRIPT'
    a.action_id = action_id
    a.key = "aw_object_id"
    a.value = params[:aw_obj_id]
  end
  action.save

  # action string
  action = QAction.new do |a|
    a.name = 'SCRIPT'
    a.action_id = action_id
    a.key = "action_string"
    a.value = params[:blockString]
  end
  action.save
end

```

```

end
action.save

# project_id
action = QAction.new do |a|
  a.name = 'SCRIPT'
  a.action_id = action_id
  a.key = "project_id"
  a.value = 1
end
action.save

# Redirect to the tweak page.
redirect_to :controller => 'objects', :action => 'tweak', :aw_obj_id => params[:aw_obj_id]
end

def scriptblocks
  @qa_object = OwnedObject.find(:first, \
                                :conditions => \
                                [ "aw_object_id = ?", params[:aw_object_id] ])
  @animations = @qa_object.palette_object.animations
end

# Ajax call to write entries to the QAction database to perform a create operation.
def create
  p = PaletteObject.find_by_model_name(params[:model_name]);

  # Create an OwnedObject for future tweaking of its parameters
  o = p.owned_objects.build
  o.aw_object_id = nil;
  o.name = p.pretty_name
  o.model = params[:model_name].to_s
  o.x = params['x'].to_i
  o.y = 0
  o.z = params['z'].to_i
  o.yaw = 0
  o.tilt = 0
  o.roll = 0
  o.project_id = 1

  o.save

  # Create a set of QActions for the ObjectEditorBot to perform the create.
  puts "Creating #{params['name']} at location(#{params['x']},#{params['z']})."

  action_id = rand(10000000)

  # model name
  action = QAction.new do |a|
    a.name = 'CREATE'
    a.action_id = action_id
    a.key = "name"
    a.value = params['model_name']
  end
  action.save

  # x coordinate
  action = QAction.new do |a|
    a.name = 'CREATE'
    a.action_id = action_id
    a.key = "x"
    a.value = params['x']
  end
  action.save

  # z coordinate
  action = QAction.new do |a|
    a.name = 'CREATE'
    a.action_id = action_id

```

```

    a.key = "z"
    a.value = params['z']
  end
  action.save

  # project_id
  action = QAction.new do |a|
    a.name = 'CREATE'
    a.action_id = action_id
    a.key = "project_id"
    a.value = 1
  end
  action.save

  render :layout => false, :json => { :status => 'success' }
end

# Ajax call to write entries to the QAction database to perform a delete operation.
def delete
  puts "Deleting object at location (#{params['x']},#{params['z']})."

  action_id = rand(10000000)

  action = QAction.new do |a|
    a.name = 'DELETE'
    a.action_id = action_id
    a.key = "x"
    a.value = params['x']
  end
  action.save

  action = QAction.new do |a|
    a.name = 'DELETE'
    a.action_id = action_id
    a.key = "z"
    a.value = params['z']
  end
  action.save

  #project_id
  action = QAction.new do |a|
    a.name = 'DELETE'
    a.action_id = action_id
    a.key = "project_id"
    a.value = 1
  end
  action.save

  render :layout => false, :json => { :status => 'success' }
end

def owned
  data = []
  @objects = Project.find(:first, \
    :conditions => \
    [ "name = ?", "Test_Project" ]).owned_objects

  @objects.each do |o|
    element = { :pretty_name => o.name.to_s, :x => o.x.to_i, :z => o.z.to_i }
    data << element
  end

  data.each do |d|
    p d
  end

  p data
  render :layout => false, :json => data
end
end

```

D Source Code – Web-App Ruby Models

D.1 animation.rb

```
class Animation < ActiveRecord::Base
  has_and_belongs_to_many :palette_objects
end
```

D.2 owned_object.rb

```
class OwnedObject < ActiveRecord::Base
  belongs_to :palette_object
  belongs_to :project
end
```

D.3 palette_object.rb

```
class PaletteObject < ActiveRecord::Base
  has_many :owned_objects
  has_and_belongs_to_many :animations
end
```

D.4 project.rb

```
class Project < ActiveRecord::Base
  has_many :owned_objects
end
```

D.5 q_action.rb

```
class QAction < ActiveRecord::Base
end
```



```

    <p>
      <%= submit_tag("Save", :class => "form_submit") %>
    </p>
  <%end%>
</body>
</html>

```

E.2 listing.html.erb

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>QA Admin Editor</title>
    <%= stylesheet_link_tag "jquery-ui-1.7.2.custom.css" %>
    <%= stylesheet_link_tag "editor.css" %>
  </head>
  <body>
    <h1>QA Models Listing</h1>
    <br>

    <table border="0" cellpadding="3" cellspacing="3">
      <tr>
        <th align="left">Name</th>
        <th align="left">Model Number</th>
        <th align="left">Description</th>
        <th align="left">Available Commands</th>
        <th align="left">Edit</th>
        <th align="left">Delete</th>
      </tr>
      <%@models.each do |m|>
        <%@avail_commands = ''%>
        <%m.animations.each do |c|>
          <%@avail_commands += c.animation%>
          <%@avail_commands += ", "%>
        <%end%>
        <%@avail_commands = @avail_commands[0, @avail_commands.length-2]%>
        <tr>
          <td><%=m.pretty_name%></td>
          <td><%=m.model_name%></td>
          <td><%=m.description%></td>
          <td><%=@avail_commands%></td>
          <td><%= link_to "Edit",
            :controller=>'admin',
            :action=>'model',
            :palette_model_name=>m.model_name%>
          </td>
          <td><%= link_to "Delete",
            :controller=>'admin',
            :action=>'delete_model',
            :palette_model_name=>m.model_name%>
          </td>
        </tr>
      <%end%>
    </table>

    <br>

    <h1>QA Commands Listing</h1>
    <br>

    <table border="0" cellpadding="3" cellspacing="3">
      <tr>
        <th align="left">Command</th>
        <th align="left">Arg Name</th>
        <th align="left">Arg Type</th>

```



```

<p>Select which commands this model can have</p>
<%@commands.each do |c|>
  <%@flag = false%>

  <%if !@model_commands.nil?>
    <%@model_commands.each do |m|>

      <%if c.animation == m.animation%>
        <%@flag = true%>
        <%break%>
      <%end%>

    <%end%>
  <%end%>

  <%if @flag%>
    <p>
      <input type="checkbox" name=<%=c.animation%> Checked/> <%=c.animation%>
    </p>
  <%else%>
    <p>
      <input type="checkbox" name=<%=c.animation%> /> <%=c.animation%>
    </p>
  <%end%>

<%end%>

<p>
  <%= submit_tag("Save", :class => "form_submit") %>
</p>
<%end%>
</body>
</html>

```

F Source Code – Web-App Ruby Object Views

F.1 edit.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
  <title>QA Object Editor</title>
  <%= stylesheet_link_tag "jquery-ui-1.7.2.custom.css" %>
  <%= stylesheet_link_tag "editor.css" %>
  <%= javascript_include_tag "jquery-1.4.1.min.js" %>
  <%= javascript_include_tag "jquery-ui-1.7.2.custom.min.js" %>
  <%= javascript_tag "var AUTH_TOKEN = #{form_authenticity_token.inspect};" if protect_against_forgery? %>
  <%= javascript_include_tag("application.js") %>
</head>
<body>
  <div id="content">
    <h1 id="mapTitle">2D Map</h1>
    " />
  </div>
  <div id="right">
    <br>
    <br>
    <%= link_to 'Object Listings', :controller => 'objects', :action => 'listing' %>
    <br>
    <h1>Palette</h1>
    <% for palette_object in @palette_objects do %>
    <p>
      <div id="<%= palette_object.model_name %>" class="draggable">
        <p><%= palette_object.pretty_name %></p>
      </div>
    </p>
    <% end %>
  </div>
</body>
</html>
```

F.2 listing.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
  <title>QA Object Editor</title>
  <%= stylesheet_link_tag "jquery-ui-1.7.2.custom.css" %>
  <%= stylesheet_link_tag "editor.css" %>
  <%= javascript_include_tag "jquery-1.4.1.min.js" %>
  <%= javascript_include_tag "jquery-ui-1.7.2.custom.min.js" %>
  <%= javascript_tag "var AUTH_TOKEN = #{form_authenticity_token.inspect};" if protect_against_forgery? %>
  <style type="text/css">
    body {
      background-color: #abebab;
    }
  </style>
</head>
<body>
<h1>Object Listings</h1>
<br>
```

```

<table border="0" cellpadding="2" cellspacing="0" width="70%">
  <tr>
    <th align="left">AW Object ID</th>
    <th align="left">Object Name</th>
    <th align="left">Description</th>
    <th align="left">X Coordinate</th>
    <th align="left">Z Coordinate</th>
    <th align="left">Click To Edit Object Properties</th>
  </tr>
  <%=created_objects.each do |o|%>
    <tr>
      <td><%=o.aw_object_id%></td>
      <td><%=o.name%></td>
      <td><%=o.descriptions%></td>
      <td><%=o.x%></td>
      <td><%=o.z%></td>
      <td><%= link_to "Edit",
                    :controller=>'objects',
                    :action=>'tweak',
                    :aw_obj_id=>o.aw_object_id%>
      </td>
    </tr>
  <%=end%>
</table>

<br>
<%= link_to 'Home', :controller => "objects", :action => "edit" %>

</body>
</html>

```

F.3 scriptblocks.html.erb

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Script Blocks</title>

    <style type="text/css">

      #blockWS {
        border: 1px solid #333333;
        background-color: #FFFFFFEE;
      }

      .label {
        margin: 0px 0px 0px 0px;
      }

      .blockPage {
        position: absolute;
        border: 1px solid #000000;
        background-color: #FEFEFE;
        width: 500px;
        height: 100%;
      }

      .blockDrawer {
        position: absolute;
        border: 1px solid #000000;
        background-color: #999999;
        width: 200px;
        height: 100%;
      }
    </style>
  </head>
  <body>
    <div id="blockWS" style="border: 1px solid #333333; background-color: #FFFFFFEE; padding: 5px;">
      <div id="blockPage" style="position: absolute; top: 0px; left: 0px; width: 500px; height: 100%; background-color: #FEFEFE;">
        <div id="blockDrawer" style="position: absolute; top: 0px; left: 0px; width: 200px; height: 100%; background-color: #999999;">
          <div id="content" style="position: absolute; top: 0px; left: 200px; width: 300px; height: 100%; padding: 5px;">
            <%=render :partial => "scriptblocks", :locals => { :scriptblocks => scriptblocks }%>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

```

        .block {
            border: 1px solid #000000;
            background-color: #9999CC;
        }
    </style>
</head>
<body>

    <div id="blockWS" style="position:absolute; left:10px; top:10px; width:700px; height:100%;
float:left">
        <div id="page" class="blockPage" style="left:200px;"></div>
    </div>

    <div id="menubar" style="position:absolute; left:720px; top:10px;">
        <div>
            <button id="clearBtn">Clear</button>
        </div>
        <div>
            <button id="drawerBtn">Next Drawer</button>
        </div>
        <div>
            <button type="button" id="saveButton">Save</button>
        </div>
    </div>

    <%= javascript_include_tag "jquery-1.4.1.min.js" %>
    <%= javascript_include_tag("script_blocks_compiled.js") %>
    <%= javascript_tag "var AUTH_TOKEN = #{form_authenticity_token.inspect};" if
protect_against_forgery? %>
    <script type="text/javascript">

        <%=animation_array = ['%>

        <%=animations.each do |a|>
            <%=animation_array += '{command: "%>
            <%=animation_array += a.animation%>
            <%=animation_array += "', arg_name: "%>
            <%=animation_array += a.arg_name%>
            <%=animation_array += "', arg_type: "%>
            <%=animation_array += a.arg_type%>
            <%=animation_array += "', '%>
        <%=end%>

        <%=animation_array = @animation_array[0, @animation_array.length - 2]%>
        <%=animation_array += ']'%>

        var commands = <%=@animation_array%>;

        var xmlStr = '<%= @qa_object.blockXML %>';

        sb.ScriptBlocksDemo.init();
        sb.ScriptBlocksDemo.run();

        $.each(commands, function(index, value){
            sb.ScriptBlocksXML.addCommandBlock(value["command"], value["arg_name"],
value["arg_type"])
        });

        sb.ScriptBlocksXML.loadXML(xmlStr);

        jQuery(function() {
            jQuery("#saveButton").click(function() {
                var form = document.createElement("form");
                form.setAttribute("method", "post");
                form.setAttribute("action", "saveScript");

                var blockString = sb.ScriptBlocksXML.getBlockStructureAsString();
                var blockXML = sb.ScriptBlocksXML.getBlockStructureAsXML();

```

```

        var blockStringForm = document.createElement("input");
        blockStringForm.setAttribute("type", "hidden");
        blockStringForm.setAttribute("name", "blockString");
        blockStringForm.setAttribute("value", blockString);

        var blockXMLForm = document.createElement("input");
        blockXMLForm.setAttribute("type", "hidden");
        blockXMLForm.setAttribute("name", "blockXML");
        blockXMLForm.setAttribute("value", blockXML);

        var awobjidField = document.createElement("input");
        awobjidField.setAttribute("type", "hidden");
        awobjidField.setAttribute("name", "aw_obj_id");
        awobjidField.setAttribute("value", <%= @qa_object.aw_object_id %>);

        var authtokenField = document.createElement("input");
        authtokenField.setAttribute("type", "hidden");
        authtokenField.setAttribute("name", "authenticity_token");
        authtokenField.setAttribute("value", AUTH_TOKEN);

        form.appendChild(blockStringForm);
        form.appendChild(blockXMLForm);
        form.appendChild(awobjidField);
        form.appendChild(authtokenField);
        document.body.appendChild(form);
        alert("About to submit.");
        form.submit();
    });
});
</script>
</body>
</html>

```

F.4 tweak.html.erb

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
  <title>QA Object Editor</title>
  <%= stylesheet_link_tag "jquery-ui-1.7.2.custom.css" %>
  <%= stylesheet_link_tag "editor.css" %>
  <%= javascript_include_tag "jquery-1.4.1.min.js" %>
  <%= javascript_include_tag "jquery-ui-1.7.2.custom.min.js" %>
  <%= javascript_tag "var AUTH_TOKEN = #{form_authenticity_token.inspect};" if protect_against_forgery? %>
  <%= javascript_include_tag("tweak.js") %>
  <%= javascript_include_tag("numeric-stepper.js") %>
  <style type="text/css">
    body {
      background-color: #abebab;
    }
  </style>
</head>
<body>
<h1>Object Options</h1>
<% form_tag({:controller => "objects", :action => "tweak"}, :method => "post") do %>
<p>
  <b>AW Object ID &nbsp;   &nbsp;  &nbsp;  &nbsp;  </b>
  <%= text_field :qa_object, :aw_object_id, :size => 20 %>
</p>

<p>

```



```

    </span>
  </span></td>

  <td><%= label :qa_object, :roll, "Roll (Z Axis): " %></td>
  <td><span class="numeric-stepper">
    <span class="rotation-stepper">
      <%= text_field :qa_object, :roll, :size => 3, :readonly => "readonly"%>
      <button type="submit" name="ns_button_1" value="1" class="plus"></button>
      <button type="submit" name="ns_button_2" value="-1" class="minus"></button>
    </span>
  </span></td>
</tr></table>
</p>

<p>
  <%= submit_tag("Save", :class => "form_submit") %>
</p>
<% end %>

<%= link_to 'Animate Object!', :controller => "objects", :action =>
"scriptblocks", :aw_object_id => @qa_object.aw_object_id%>

<br>

<%= link_to 'Back to Object Listing', :controller => 'objects', :action => 'listing' %>

</body>
</html>

```


G Source Code – Javascript

G.1 numeric-stepper.js

```
/**
 * Numeric Stepper
 * -----
 *
 * Copyright 2007 Ca Phun Ung
 *
 * This software is licensed under the CC-GNU LGPL
 * http://creativecommons.org/licenses/LGPL/2.1/
 *
 * Version 0.1
 */

/**
 * Numeric Stepper Class.
 */
var RotationNumericStepper = {
  register : function(name, minValue, maxValue, stepSize){
    this.minValue = minValue;
    this.maxValue = maxValue;
    this.stepSize = stepSize;
    var elements = getElementsByClassName(document, "*", name);
    for (var i=0; i<elements.length; i++){
      var textbox = elements[i].getElementsByTagName('input')[0];
      if (textbox){
        if (textbox.value == undefined || textbox.value == '' || isNaN(textbox.value))
          textbox.value = 0;
        textbox.onkeypress = function(e){
          if(window.event){
            keynum = e.keyCode; // IE
          } else if(e.which){
            keynum = e.which; // Netscape/Firefox/Opera
          }
          keychar = String.fromCharCode(keynum);
          numcheck = /^[0-9\-]/;
          if (keynum==8)
            return true;
          else
            return numcheck.test(keychar);
        };
        textbox.onblur = function(){
          if (parseInt(this.value) < RotationNumericStepper.minValue)
            this.value = RotationNumericStepper.minValue;
          if (parseInt(this.value) > RotationNumericStepper.maxValue)
            this.value = RotationNumericStepper.maxValue;
        };
        var buttons = elements[i].getElementsByTagName('button');
        if (buttons[0]){
          this.addButtonEvent(buttons[0], textbox, this.stepUp);
        }
        if (buttons[1])
          this.addButtonEvent(buttons[1], textbox, this.stepDown);
      }
    }
  },
  addButtonEvent:function(o,textbox, func){
    o.textbox = textbox;
    o.onclick = func;
  },
  stepUp:function(){
    RotationNumericStepper.stepper(this.textbox, RotationNumericStepper.stepSize);
  }
}
```

```

,stepDown:function(){
  RotationNumericStepper.stepper(this.textbox, -RotationNumericStepper.stepSize);
}
,stepper:function(textbox, val){
  if (textbox == undefined)
    return false;
  if (val == undefined || isNaN(val))
    val = 1;
  if (textbox.value == undefined || textbox.value == '' || isNaN(textbox.value))
    textbox.value = 0;
  textbox.value = parseInt(textbox.value) + parseInt(val);
  if (parseInt(textbox.value) < RotationNumericStepper.minValue)
    textbox.value = RotationNumericStepper.minValue;
  if (parseInt(textbox.value) > RotationNumericStepper.maxValue)
    textbox.value = RotationNumericStepper.maxValue;
}
}
}

var LocationNumericStepper = {
  register : function(name, minValue, maxValue, stepSize){
    this.minValue = minValue;
    this.maxValue = maxValue;
    this.stepSize = stepSize;
    var elements = getElementsByClassName(document, "*", name);
    for (var i=0; i<elements.length; i++){
      var textbox = elements[i].getElementsByTagName('input')[0];
      if (textbox){
        if (textbox.value == undefined || textbox.value == '' || isNaN(textbox.value))
          textbox.value = 0;
        textbox.onkeypress = function(e){
          if(window.event){
            keynum = e.keyCode; // IE
          } else if(e.which){
            keynum = e.which; // Netscape/Firefox/Opera
          }
          keychar = String.fromCharCode(keynum);
          numcheck = /[0-9\-\-]/;
          if (keynum==8)
            return true;
          else
            return numcheck.test(keychar);
        };
        textbox.onblur = function(){
          if (parseInt(this.value) < LocationNumericStepper.minValue)
            this.value = LocationNumericStepper.minValue;
          if (parseInt(this.value) > LocationNumericStepper.maxValue)
            this.value = LocationNumericStepper.maxValue;
        };
        var buttons = elements[i].getElementsByTagName('button');
        if (buttons[0]){
          this.addButtonEvent(buttons[0], textbox, this.stepUp);
        }
        if (buttons[1])
          this.addButtonEvent(buttons[1], textbox, this.stepDown);
      }
    }
  }
,addButtonEvent:function(o,textbox, func){
  o.textbox = textbox;
  o.onclick = func;
}
,stepUp:function(){
  LocationNumericStepper.stepper(this.textbox, LocationNumericStepper.stepSize);
}
,stepDown:function(){
  LocationNumericStepper.stepper(this.textbox, -LocationNumericStepper.stepSize);
}
,stepper:function(textbox, val){
  if (textbox == undefined)

```

```

        return false;
    if (val == undefined || isNaN(val))
        val = 1;
    if (textbox.value == undefined || textbox.value == '' || isNaN(textbox.value))
        textbox.value = 0;
    textbox.value = parseInt(textbox.value) + parseInt(val);
    if (parseInt(textbox.value) < LocationNumericStepper.minValue)
        textbox.value = LocationNumericStepper.minValue;
    if (parseInt(textbox.value) > LocationNumericStepper.maxValue)
        textbox.value = LocationNumericStepper.maxValue;
    }
}

/**
 * getElementsByClassName - returns an array of elements selected by their class name.
 * @author Jonathan Snook <http://www.snook.ca/jonathan>
 * @add-ons Robert Nyman <http://www.robertnyman.com>
 */
function getElementsByClassName(oElm, strTagName, strClassName){
    var arrElements = (strTagName == "*" && oElm.all)? oElm.all : oElm.getElementsByTagName(strTagName);
    var arrReturnElements = new Array();
    strClassName = strClassName.replace(/-/g, "\\-");
    var oRegExp = new RegExp("(^|\\s)" + strClassName + "(\\s|$)");
    var oElement;
    for(var i=0; i<arrElements.length; i++){
        oElement = arrElements[i];
        if(oRegExp.test(oElement.className)){
            arrReturnElements.push(oElement);
        }
    }
    return (arrReturnElements)
}

function initNumericStepper(){
    var myLocationNumericStepper = LocationNumericStepper.register("location-stepper", 0, 10, 1);
    var myRotationNumericStepper = RotationNumericStepper.register("rotation-stepper", 0, 180, 1);
}

/**
 * addEvent - simple window.onload event loader.
 */
function addEvent(o, evt, f){
    var r = false;
    if (o.addEventListener){
        o.addEventListener(evt, f, false);
        r = true;
    }
    else if (o.attachEvent)
        r = o.attachEvent("on"+evt, f);
    return r;
}
addEvent(window, "load", initNumericStepper);

```

G.2 ScriptblocksXML.js

```

goog.provide('sb.ScriptBlocksXML');

goog.require('sb.ScriptBlocks');
goog.require('sb.Block');
goog.require('sb.BlockSpec');
goog.require('sb.Drawer');
goog.require('sb.Page');
goog.require('sb.Workspace');

/**
 * ScriptBlocksXML.js - a convenience class with static methods for initializing,
 * loading, and saving ScriptBlocks workspaces from and to XML.
 */

```

```

* @author djwendel
*
* Modified ScriptBlocksXML.js to create the workspace needed for Quest Atlantis. Instead
* of initializing and loading the workspace from a XML file, the specification for the Quest
* Atlantis workspace is inputted via the createblock function.
*
* @author David Lam
*/

sb.ScriptBlocksXML.commands = []; //array consisting of all the commands that were added to this workspace

sb.ScriptBlocksXML.genusMap = {}; //maps genus name to BlockSpec for that genus

sb.ScriptBlocksXML.loadXML = function(xmlStr){
    sb.ScriptBlocksXML.createWorkSpace();
    sb.ScriptBlocksXML.loadBlockStructureFromXML(xmlStr);
}

sb.ScriptBlocksXML.parseColor = function(colorStr){
    return 0x888822;
}

/**
 * Create a block given the specified attributes of the block
 *
 * @param {Object} blockName
 * @param {Object} blockLabel
 * @param {Object} blockInitLabel
 * @param {Object} blockReturnType
 * @param {Object} blockArguments
 * @param {Object} blockLangSpecProperties
 * @param {Object} blockColor
 * @param {Object} blockDrawer - string indicating which drawer this block belongs to
 * @param {Object} isACommand - boolean indicating if the block is a command block
 */
sb.ScriptBlocksXML.createBlock = function(blockName, blockLabel, blockInitLabel, blockReturnType,
blockArguments,
                                blockLangSpecProperties, blockColor, blockDrawer, isACommand){

    var spec = {};

    spec = new sb.BlockSpec();
    spec = sb.BlockSpec.extend(spec, {
        name: blockName,
        label: blockLabel,
        initLabel: blockInitLabel,
        returnType: blockReturnType,
        arguments: blockArguments,
        langSpecProperties: blockLangSpecProperties,
        color: blockColor,
        drawer: blockDrawer
    });

    if(isACommand){
        spec = sb.BlockSpec.extend(spec, {
            connections: [sb.SocketType.AFTER, sb.SocketType.BEFORE]
        });
    } else {
        spec = sb.BlockSpec.extend(spec, {
            connections: [sb.SocketType.BEFORE]
        });
    }

    // save spec to the genus spec map
    sb.ScriptBlocksXML.genusMap[blockName] = spec;
}

/**

```

```

* Create the trigger blocks
*/
sb.ScriptBlocksXML.createTriggerBlocks = function(){

    var spec = {};
    var specArgs;
    var specLSPs;

    //Create WhenClicked Block
    specArgs = [];
    specArgs.push({
        name: "actions",
        type: "cmd",
        socketType: "nested"
    });

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "WhenClicked";

    sb.ScriptBlocksXML.createBlock("WhenClicked", "WhenClicked @actions\n", "WhenClicked", "command",
specArgs, specLSPs, "255 0 0", "trigger", true);

    //Create WhenBumped Block
    specArgs = [];
    specArgs.push({
        name: "actions",
        type: "cmd",
        socketType: "nested"
    });

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "WhenBumped";

    sb.ScriptBlocksXML.createBlock("WhenBumped", "WhenBumped @actions\n", "WhenBumped", "command",
specArgs, specLSPs, "255 0 0", "trigger", true);

    //Create WhenApproached Command Block
    specArgs = [];
    specArgs.push({
        name: "actions",
        type: "cmd",
        socketType: "nested"
    });

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "WhenApproached";

    sb.ScriptBlocksXML.createBlock("WhenApproached", "WhenApproached @actions\n", "WhenApproached",
"command", specArgs, specLSPs, "255 0 0", "trigger", true);

    //Create RightAway Command Block
    specArgs = [];
    specArgs.push({
        name: "actions",
        type: "cmd",
        socketType: "nested"
    });

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "RightAway";

    sb.ScriptBlocksXML.createBlock("RightAway", "RightAway @actions\n", "RightAway", "command", specArgs,
specLSPs, "255 0 0", "trigger", true);

}

/**
* Create a command block based on the given specs
*/

```

```

sb.ScriptBlocksXML.addCommandBlocks = function(command, arg_name, arg_type){

    var specArgs;
    var specLSPs;

    specArgs = [];
    specArgs.push({
        name: arg_name,
        type: arg_type,
        socketType: "nested"
    });

    specLSPs = {};
    specLSPs["vm-cmd-name"] = command;
    specLSPs["stack-type"] = "breed";

    sb.ScriptBlocksXML.createBlock(command, command + " @" + arg_name + "\n", command, "command", specArgs,
specLSPs, "255 0 0", "command", true);

    sb.ScriptBlocksXML.commands.push(command);
}

/**
 * Create the constant blocks
 */
sb.ScriptBlocksXML.createConstantBlocks = function(){
    var spec = {};
    var specArgs;
    var specLSPs;

    //Create Number Block with a constant of 1
    specArgs = [];

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "eval-num";
    specLSPs["is-monitorable"] = "yes";

    sb.ScriptBlocksXML.createBlock("One", "1 ", "1", "number", specArgs, specLSPs, "255 0 0", "constant",
false);

    //Create Number Block with a constant of 5
    specArgs = [];

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "eval-num";
    specLSPs["is-monitorable"] = "yes";

    sb.ScriptBlocksXML.createBlock("Five", "5 ", "5", "number", specArgs, specLSPs, "255 0 0", "constant",
false);

    //Create Number Block with a constant of 10
    specArgs = [];

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "eval-num";
    specLSPs["is-monitorable"] = "yes";

    sb.ScriptBlocksXML.createBlock("Ten", "10 ", "10", "number", specArgs, specLSPs, "255 0 0", "constant",
false);

    //Create String block with a constant this is a test string
    specArgs = [];

    specLSPs = {};
    specLSPs["vm-cmd-name"] = "eval-num";
    specLSPs["is-monitorable"] = "yes";

    sb.ScriptBlocksXML.createBlock("TestString", "This is a test string ", "This is a test string",
"string", specArgs, specLSPs, "255 0 0", "constant", false);

```

```

}

/**
 * Create the trigger drawers which consist of all the trigger blocks
 */
sb.ScriptBlocksXML.createTriggerDrawer = function(){
    var drawer;
    var block;

    drawer = new sb.Drawer("Triggers");
    sb.ScriptBlocksDemo.workspace.addDrawer(drawer);

    if(sb.ScriptBlocksXML.genusMap["WhenClicked"] == undefined){
        console.log("No genus definition for WhenClicked");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["WhenClicked"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["WhenBumped"] == undefined){
        console.log("No genus definition for WhenBumped");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["WhenBumped"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["WhenApproached"] == undefined){
        console.log("No genus definition for WhenApproached");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["WhenApproached"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["RightAway"] == undefined){
        console.log("No genus definition for RightAway");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["RightAway"]);
        drawer.addBlock(block);
    }
}

/**
 * Create the command drawer which consist of all the command blocks
 */
sb.ScriptBlocksXML.createCommandDrawer = function(){
    var drawer;
    var block;

    drawer = new sb.Drawer("Commands");
    sb.ScriptBlocksDemo.workspace.addDrawer(drawer);

    for(i = 0; i < sb.ScriptBlocksXML.commands.length; i++){

        if(sb.ScriptBlocksXML.genusMap[sb.ScriptBlocksXML.commands[i]] == undefined){
            console.log("No genus definition for " + sb.ScriptBlocksXML.commands[i]);
        }
        else{
            block = new sb.Block(sb.ScriptBlocksXML.genusMap[sb.ScriptBlocksXML.commands[i]]);
            drawer.addBlock(block);
        }
    }
}

/**
 * Create the constant drawer which consists fo all the constant blocks

```

```

*/
sb.ScriptBlocksXML.createConstantDrawer = function(){
    var drawer;
    var block;

    drawer = new sb.Drawer("Constants");
    sb.ScriptBlocksDemo.workspace.addDrawer(drawer);

    if(sb.ScriptBlocksXML.genusMap["One"] == undefined){
        console.log("No genus definition for One");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["One"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["Five"] == undefined){
        console.log("No genus definition for Five");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["Five"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["Ten"] == undefined){
        console.log("No genus definition for Ten");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["Ten"]);
        drawer.addBlock(block);
    }

    if(sb.ScriptBlocksXML.genusMap["TestString"] == undefined){
        console.log("No genus definition for TestString");
    }
    else{
        block = new sb.Block(sb.ScriptBlocksXML.genusMap["TestString"]);
        drawer.addBlock(block);
    }
}

/**
 * Create the workspace by creating all the needed blocks
 * and drawers
 */
sb.ScriptBlocksXML.createWorkSpace = function(){

    //Create all the needed blocks
    sb.ScriptBlocksXML.createTriggerBlocks();
    sb.ScriptBlocksXML.createConstantBlocks();

    //Create all the needed drawers
    sb.ScriptBlocksXML.createTriggerDrawer();
    sb.ScriptBlocksXML.createCommandDrawer();
    sb.ScriptBlocksXML.createConstantDrawer();
}

/**
 * loads a "project" - a set of blocks which are potentially interconnected, from saved XML
 * @param {Object} xml
 */
sb.ScriptBlocksXML.loadBlockStructureFromXML = function(xml){
    var parser = new DOMParser();
    var xmlDoc = parser.parseFromString(xml,"text/xml");
    var blockElts = xmlDoc.getElementsByTagName("Block");
    var spec;
    var blocksByID = {};

    for (var i=0; i<blockElts.length; i++) {

```



```

spec = new sb.BlockSpec();
spec = sb.BlockSpec.extend(spec, sb.ScriptBlocksXML.genusMap[blockElts[i].getAttribute("genus-
name")]));

var connectorElts = blockElts[i].getElementsByTagName("BlockConnector");

var specLabel = spec.initLabel+' ';
var specInitLabel = spec.initLabel;
if (blockElts[i].getElementsByTagName("Label").length > 0) {
    specInitLabel = blockElts[i].getElementsByTagName("Label")[0].textContent;
    specLabel = specInitLabel+' ';
}

var specReturnType = spec.returnType;
var blockID = blockElts[i].getAttribute("id");

// parse the arguments (sockets) for this instance of the block
var specArgs = [];
for (var j=0; j<connectorElts.length; j++) {
    if (connectorElts[j].getAttribute("connector-kind") != "socket") {
        specReturnType = connectorElts[j].getAttribute("connector-type");
        continue; // only use sockets, not plugs, to create arguments spec
    }
    if (connectorElts[j].hasAttribute("label") && connectorElts[j].getAttribute("label") !=
'' ) {

        argLabel = connectorElts[j].getAttribute("label");
        //specLabel += argLabel;
    } else {
        argLabel = 'arg'+j;
    }
    specLabel += '@'+argLabel+'\n';
    specArgs.push({
        name: argLabel,
        type: connectorElts[j].getAttribute("connector-type"),
        socketType: "nested"
    });
}
if (specArgs.length == 0) { //if no arg override happened, use default args from the genus
specArgs = spec.arguments;
}

// parse LangSpecProperties
lspElts = blockElts[i].getElementsByTagName("LangSpecProperty");
specLSPs = spec.langSpecProperties;
for (var k=0; k<lspElts.length; k++) {
    specLSPs[lspElts[k].getAttribute("key")] = lspElts[k].getAttribute("value");
}

// update the BlockSpec for this instance of the block
spec = sb.BlockSpec.extend(spec, {
    label: specLabel,
    initLabel: specInitLabel,
    returnType: specReturnType,
    arguments: specArgs,
    langSpecProperties: specLSPs
});

// ID-block mapping (for connection next)
var block = new sb.Block( spec );
blocksByID[blockID] = block;
}

// now loop through and make all of the required connections
for (i = 0; i < blockElts.length; i++) {

    blockID = blockElts[i].getAttribute("id");
    connectorElts = blockElts[i].getElementsByTagName("BlockConnector");

```

```

// look through the sockets and make connections if found
for (var j = 0; j < connectorElts.length; j++) {
    if (connectorElts[j].getAttribute("connector-kind") != "socket") {
        continue; // only connect sockets blocks
    }
    var conBlockID = connectorElts[j].getAttribute("con-block-id");
    var argName;
    if (conBlockID != '' && conBlockID != undefined) {
        if (connectorElts[j].hasAttribute("label") &&
connectorElts[j].getAttribute("label") != '') {
            argName = connectorElts[j].getAttribute("label");
        } else {
            argName = 'arg'+j;
        }
        // connect the child block (ID = conBlockID) to this block in this socket
        blocksByID[blockID].connectChildBlock(blocksByID[conBlockID], argName);
    }
}

// now connect the block below it, if one exists
if (blockElts[i].getElementsByName("AfterBlockId").length > 0) {

    blocksByID[blockID].connectAfterBlock(blocksByID[blockElts[i].getElementsByName("AfterBlockId")[0]
.textContent]);
}

}

// add the top-level blocks to the page
for (blockID in blocksByID) {
    var topBlock = blocksByID[blockID];

    if(topBlock.getParent() === null && topBlock.getBefore() === null) {
        sb.ScriptBlocksDemo.testPage.addBlock( topBlock );
    }
}
}

/**
 * returns an XML string representation of the block structure on the page.
 * @return {string}
 */
sb.ScriptBlocksXML.getBlockStructureAsXML = function(){
    var pages = sb.ScriptBlocks.getWorkspace().getPages();
    var xmlstr = '<?xml version="1.0" encoding="UTF-16"?><SLCODEBLOCKS><PageBlocks>';
    var idcounter = 1; //non-zero ID's just to be safe

    // first, assign every Block an ID to make linking much easier
    for (var i = 0; i < pages.length; i++) {
        var pageBlocks = pages[i].getAllBlocks();
        for (var j = 0; j < pageBlocks.length; j++) {
            pageBlocks[j].id = idcounter;
            //console.log(idcounter + ": " + pageBlocks[j].getSpec().initLabel + ": " +
pageBlocks[j].getSpec().drawer);
            idcounter++;
        }
    }

    // now go through again and generate XML for each block
    for (i=0; i<pages.length; i++) {
        pageBlocks = pages[i].getAllBlocks();
        for (j=0; j<pageBlocks.length; j++) {
            xmlstr += '<Block ';
            xmlstr += 'id="' + pageBlocks[j].id + '" ';
            xmlstr += 'genus-name="' + pageBlocks[j].getSpec().name + '">';
            xmlstr += '<Label>' + pageBlocks[j].getSpec().initLabel + '</Label>';
            if (pageBlocks[j].getSpec().returnType != "command") {
                xmlstr += '<Plug><BlockConnector connector-kind="plug" connector-type="' +
pageBlocks[j].getSpec().returnType + '" ';
                xmlstr += 'init-type="' + pageBlocks[j].getSpec().returnType + '" label="'

```

```

position-type="mirror" ';
        if (pageBlocks[j].getParent() instanceof sb.Block) {
            xmlstr += 'con-block-id="' + pageBlocks[j].getParent().id + '" ';
        }
        xmlstr += '></BlockConnector></Plug>';
    }
    //if (pageBlocks[j].getArguments().length > 0) { // TODO:NO WAY TO TEST FOR # of ARGS??
    xmlstr += '<Sockets num-sockets="' + 1 /*TODO:WRONG!*/ + '" >';
    for (var argName in pageBlocks[j].getArguments()) {
        var arg = pageBlocks[j].getArguments()[argName];
        xmlstr += '<BlockConnector connector-kind="socket" ';
        xmlstr += 'connector-type="' + arg.getDataType() + '" ';
        xmlstr += 'init-type="' + arg.getDataType() + '" ';
        xmlstr += 'label="' + argName + '" ';
        if (arg.getValue() instanceof sb.Block) {
            xmlstr += 'con-block-id="' + arg.getValue().id + '" ';
        }
        xmlstr += '></BlockConnector>';
    }
    xmlstr += '</Sockets>';
    //}
    if (pageBlocks[j].getAfterBlock() instanceof sb.Block) {
        xmlstr += '<AfterBlockId>' + pageBlocks[j].getAfterBlock().id +
'</AfterBlockId>';
    }
    xmlstr += '</Block>';
}
}
xmlstr += '</PageBlocks></SLCODEBLOCKS>';
return xmlstr;
}

/**
 * Given the current block structure in the workspace,
 * return a string representation that can be parsed by the
 * Quest Atlantis bot.
 */
sb.ScriptBlocksXML.getBlockStructureAsString = function(){
    var pages = sb.ScriptBlocks.getWorkspace().getPages();
    var retStr = '';
    var idcounter = 1;
    var lastTriggerID = 0;

    // first, assign every Block an ID to make linking much easier
    for (var i = 0; i < pages.length; i++) {
        var pageBlocks = pages[i].getAllBlocks();
        for (var j = 0; j < pageBlocks.length; j++) {
            pageBlocks[j].id = idcounter;
            idcounter++;
        }
    }

    // now start generating the string representation of the block structure
    for (i = 0; i < pages.length; i++){
        pageBlocks = pages[i].getAllBlocks();
        retStr += "{";
        while(lastTriggerID < pageBlocks.length){
            //Find the next trigger block
            if(pageBlocks[lastTriggerID].getSpec().drawer == "trigger"){
                retStr += pageBlocks[lastTriggerID].getSpec().initLabel + "[";

                var done = false;
                var lastCommandBlockID = 0;

                //Find the corresponding initial command block
                for(var argName1 in pageBlocks[lastTriggerID].getArguments()){
                    var triggerArg = pageBlocks[lastTriggerID].getArguments()[argName1];

                    if(triggerArg.getValue() instanceof sb.Block){

```

```

var commandBlockID = triggerArg.getValue().id;
lastCommandBlockID = commandBlockID;

if(pageBlocks[commandBlockID - 1].getSpec().drawer == "command"){
    retStr += pageBlocks[commandBlockID - 1].getSpec().initLabel + "(";

    //Find the corresponding constant block
    for(var argName2 in pageBlocks[commandBlockID - 1].getArguments()){
        var commandArg = pageBlocks[commandBlockID - 1].getArguments()[argName2];

        if(commandArg.getValue() instanceof sb.Block){
            var constantBlockID = commandArg.getValue().id;

            if(pageBlocks[constantBlockID - 1].getSpec().drawer == "constant"){
                retStr += pageBlocks[constantBlockID - 1].getSpec().initLabel +
                " ";
            }
        }
    }
}

//Find the command blocks following the initial command block
while(!done){
    if (pageBlocks[lastCommandBlockID-1] instanceof sb.Block &&
        pageBlocks[lastCommandBlockID - 1].getAfterBlock() instanceof sb.Block){

        var commandBlockID = pageBlocks[lastCommandBlockID - 1].getAfterBlock().id;

        if(pageBlocks[commandBlockID - 1].getSpec().drawer == "command"){
            retStr += "," + pageBlocks[commandBlockID - 1].getSpec().initLabel + "(";

            //Find the corresponding constant block
            for(var argName2 in pageBlocks[commandBlockID - 1].getArguments()){
                var commandArg = pageBlocks[commandBlockID - 1].getArguments()[argName2];
                var constantBlockID = commandArg.getValue().id;

                if(pageBlocks[constantBlockID - 1].getSpec().drawer == "constant"){
                    retStr += pageBlocks[constantBlockID - 1].getSpec().initLabel + " ";
                }
            }

            lastCommandBlockID = commandBlockID;
        }
    }
    else{
        done = true;
    }
}
retStr += "], ";
lastTriggerID++;
}

lastTriggerID = 0;
retStr = retStr.substring(0, retStr.length - 2);
retStr += "}";
}

return retStr;
}

```