Pet Park: A Virtual Learning World for Kids

by

Austina M. De Bonte

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 21, 1998

Author________________________________
Department of Electrical Engineering and Computer Science
May 21, 1998

Certified by________________________________
Mitchel Resnick
Thesis Supervisor

Accepted by________________________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Pet Park: A Virtual Learning World for Kids

Austina M. De Bonte

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

May 21, 1998

Thesis Supervisor: Mitchel Resnick, MIT Media Laboratory

Abstract

Pet Park is a graphical, networked virtual world for kids. Kids can log in to Pet Park over the Internet to talk with other kids and create new pets. They can design behaviors for their pets by writing scripts with a simple programming language for kids called YoYo. Pet Park is a collaborative building environment that recognizes the importance of community support for construction. This thesis discusses the differences between graphical online worlds like Pet Park and their text-based cousins. Many of these differences have important ramifications for the design and implementation of constructionist graphical environments. This thesis goes on to present a set of design criteria for a next generation of graphical learning worlds.

Acknowledgements

I would like to extend thanks to my advisor, Mitchel Resnick, for contributing, shaping, and focusing these ideas into a yearlong project, and for being a gentle and wise advisor. I would also like to thank Amy Bruckman for taking me under her wing three years ago. Amy has been a terrific mentor for me. Her work has inspired many of the ideas in this thesis. MOOSE Crossing made the idea of constructionism come alive in my mind. Thanks to Andy Begel for creating and supporting my use of YoYo, without which the full scope of this project would not be possible. Thanks to Brian Smith for being there to share in the excitement, and for offering a different viewpoint. Thanks also to my numerous colleagues and friends in the Epistemology and Learning Group who have always been willing to lend an ear, suggest an idea, or offer moral support, including Rick Borovoy, David Cavallo, Vanessa Colella, Chris Hancock, Kwin Kramer, Fred Martin, Bakhtiar Mikhak, Marina Umashi-Bers, and David Shaffer. Many thanks to Brian Silverman for always challenging my assumptions, for his wise comments, and for our thought provoking discussions. A special thanks to Florence Williams, for everything she does to help keep things running smoothly.

A great big thanks to Chris Genly without whose patient assistance and well-timed visit I would still be wondering how the MU SDK really works. Thanks also to Scott Lewis for bouncing around early ideas of Pet Park, and for introducing me to the MU SDK last summer.

Thanks to Jaime Meritt for his great work on the Pet Park client, to Veronica Weiner for taking on the help system, and to Andrew Cheng, for seeking me out to work on the graphical programming environment that I would never have gotten to on my own.

Thanks also to Mark Loughridge and others of ImagineEngine Corp. for contributing their superior artistic talents and great ideas.

Thank you to all the kids and adults around the world who have tried out Pet Park, and especially those who have given me feedback about their experiences. And an extra big thank you to the participants of the afterschool program, and to their parents.

To my parents, sister, family, and friends: thank you for your love and support throughout my years at MIT, and always. I couldn't have done it without you. From the bottom of my heart, thank you.

Finally, I send a very special thanks to my husband, Erik.

Table of Contents

1 Introduction

Pet Park is a graphical, online world for kids. Kids can log onto Pet Park over the Internet to explore the virtual world, talk to other kids, and make new pets. They can create new objects to populate the world, such as a dog that waves whenever its owner enters the room, or a chair that says "Ouch!" whenever anyone sits in it. Kids teach their creatures behaviors using a special kid-friendly programming language called YoYo (Begel 1997). As kids build in Pet Park, they share their projects with other kids online, and get help from peers when they get stuck. Pet Park is a collaborative building environment that enables and encourages kids to design creatures in the virtual world.

Pet Park was inspired by a project called MOOSE Crossing. Created by Amy Bruckman, MOOSE Crossing is a text-based virtual world for kids (Bruckman 1997). At MOOSE Crossing, kids can create anything they can describe in words, write programs for their creations, and play with other kids' creations. MOOSE Crossing is fundamentally based on the idea of kids building a virtual world collaboratively, learning and supporting each other along the way. I worked alongside Amy Bruckman on MOOSE Crossing for two years, working with kids online, helping kids with their projects, and refining the design of the MOOSE Crossing virtual world. Using MOOSE Crossing as a model, I wanted Pet Park to be a building space for kids with a vibrant, online community, but situated in a graphical environment rather than a textual one.

Designing Pet Park as a graphical online world was a big change from the text-based MOOSE Crossing world. Simply adding graphics to MOOSE Crossing was not only impractical, but also failed to take advantage of the affordances graphics had to offer to this kind of virtual world. The Pet Park design borrowed many ideas from MOOSE

Crossing, but in the end, was a new, experimental design in how graphics can be used to enable and enhance a construction-oriented world.

The Pet Park design discussed in this thesis does not pay a lot of attention to the design of the particular programming language kids use to program their creatures. Andrew Cheng has been studying the programming language for Pet Park, and has designed a graphical front end to the YoYo language used in the Pet Park prototype (Cheng 1998). The issue of language design for a graphical online world such as Pet Park is an important one, but was not studied in the context of this thesis due to time constraints.

1.1 Thesis Overview

This thesis has five main chapters. This chapter describes the motivation and background for this project, and gives a brief survey of the Pet Park and MOOSE Crossing virtual worlds. Chapter 2 goes on to present the initial vision of Pet Park in more detail. It then describes the prototype Pet Park system currently available over the Internet, and presents a sampling of projects that kids have created in Pet Park. Chapter 3 takes a hard look at the differences between text and graphics as primary media for a construction-oriented virtual world. Using MOOSE Crossing and Pet Park as examples, it analyses the pros and cons of using text and graphics, and discusses the ways in which a graphics-based world like Pet Park is fundamentally different from a text-based world like MOOSE Crossing. Many of these differences have important ramifications for the design and implementation of constructionist graphical environments. Chapter 4 presents a set of criteria for designing graphical, constructionist online worlds, influenced both by the discussion in Chapter 3 and from practical experience working with the Pet Park prototype. Chapter 5 concludes with a plethora of new questions and new directions for this branch of research.

1.2 MOOSE Crossing

In the virtual world called MOOSE Crossing, kids create pet penguins, magic subways, and lush green jungles (Bruckman 1997). They write elaborate descriptions of the things and places they build, and then write computer programs to make them come alive. Dogs jump up to kiss you on the nose, chairs smile when you sit in them, and rooms announce to their owners who has visited recently. MOOSE Crossing is a place where kids can build whatever they can describe in words.

Amy Bruckman developed MOOSE Crossing as a place for kids ages eight to thirteen to do reading, writing, and computer programming in a supportive community environment. At MOOSE Crossing, everyone is engaged in a common activity—learning how to create things in the virtual world. Kids are not just learners at MOOSE Crossing, they are teachers as well. Older members help younger ones with their programming projects, and everyone, even the youngest child, helps new users learn how to get around and build their first object—most often a pet dog. Kids share their creations with their peers, getting positive feedback and basking in the glow of an appreciative audience. Peers help each other with their projects, supporting each other as they learn, and encouraging each other to build ever larger and more complex projects. Kids and adults who develop larger skill sets act as role models for other members, motivating them to achieve a similar level of expertise.

Kids' projects on MOOSE Crossing range from objects with simple one-line scripts to complex, multipurpose rooms with several dozen scripts, totaling two hundred lines of code or more. A thirteen-year-old girl made a beautiful lake where other kids can make boats, sail them around the lake, and build their own houses on one of two islands. A nine-year-old boy made a shop where you can buy a newt, with help from a friendly shopkeeper. An eleven-year-old girl made a pet penguin that you can feed, hug, and kiss, among other things. A twelve-year-old boy made a swimming pool where kids can swim, dive, and go underwater. A class of fifth and sixth graders built an elaborate cruise ship that sinks like the Titanic, totaling more than 20 interconnected rooms.

Kids' projects on MOOSE Crossing make use of sophisticated concepts in computer programming. Projects use local variables and object state variables, called properties, to keep track of data, such as whether a pet pig is currently on a diet. Forking off a task for later execution is commonly used to have scripts wait before doing something, such as a penguin that waits a few minutes after it's fed before asking for more food. Kids use control structures in their scripts, such as if-then-else statements and for loops. More advanced projects use arrays, sometimes even using several arrays in parallel for more extensive data storage.

1.3 Constructionism

MOOSE Crossing is fundamentally based on the idea of constructionism, a theory of learning developed by Seymour Papert. Constructionism is an extension of Jean Piaget's theory of constructivism. Papert explains constructionism this way:

> Constructionism—the N word as opposed to the V word—shares constructivism's connotation of learning as "building knowledge structures" irrespective of the circumstances of learning. It then adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe (Papert 1991).

Constructionism emphasizes that these projects should be personally meaningful, allowing the learner to make deep personal connections with the ideas being learned (Papert 1991, Resnick 1994). Bruckman's work with MOOSE Crossing expands the idea of constructionism by recognizing the importance of community support for constructionist learning activities (Bruckman 1997). Bruckman explores the bi-directional link between constructionism and community, finding both that communities are vitally important for supporting constructionist learning activities, and that constructionist activities, such as collaboratively building a virtual world, shape, strengthen, and enhance the community that surrounds those activities.

In the case of MOOSE Crossing, interacting with a community of peers helps kids learn how to create new things and figure out how to write their own programs. The community motivates and supports kids' learning experiences by providing a plethora of examples. Plenty of individuals are around, eager to share their knowledge and help work through problems. Conversely, the fact that MOOSE Crossing is a place specifically designed for building greatly influences the kind of community and the culture that develops there. Individuals who are skilled builders earn respect and admiration. Having a common goal gives members of the community something substantive to talk about, and an endless stream of projects they can work on together to construct. Bruckman argues that communities like MOOSE Crossing greatly benefit from this kind of "constructionist culture" (Bruckman 1997).

1.4 Programming as a Constructionist Learning Activity

Papert argues that computers can be excellent materials for constructionist learning activities (Papert 1980). To explore this idea, Papert developed the Logo programming language, which allows kids to explore planar geometry by programming a graphical "turtle" to move around a computer screen. He found that kids both developed intuitions about geometry and built strong relationships to powerful mathematical ideas such as algorithms, variables, and procedures through working on their Logo projects.

When writing a Logo computer program, kids must externalize their thoughts into a specific sequence of instructions. An important result of writing the thought process down in this way is that the child can go back and re-examine the program, modifying it if necessary. The child's thought process becomes more concrete and explicit, and hence, more accessible to reflection (Papert 1980).

By working on Logo projects, kids learn to manage complexity in their projects with procedures and subprocedures, breaking their programs up into "mind-sized bites." Papert argues that this kind of "procedural thinking is a powerful intellectual tool," helping kids learn to think through problems step-by-step (Papert 1980).

Papert argues vehemently against the conventional model of learning drilled in school. In a traditional school setting, "many children are held back in their learning because they have a model of learning in which you have either 'got it' or 'got it wrong.'" Papert argues that teaching kids how to program can help break this trend. Programmers, both young and old, rarely write their programs correctly the first time. Rather, learning to be a good programmer means learning how to debug programs, isolating problems and fixing them. Programming is a good way to learn debugging strategies. Kids can then apply these strategies and this new debugging mindset to other learning activities (Papert 1980).

1.5 Layout of Multi-User Online Environments

Pet Park and MOOSE Crossing are examples of a much broader class of multi-user, online environments. These virtual environments can use text, 2-dimensional graphics, 3-dimensional graphics, or some hybrid of the three as their delivery media. However, most of these environments, regardless of media, typically share certain features in layout and structure.

Human users that join these worlds are called players, members, characters, or avatars. The term avatar can also refer to the graphical depiction of a player's character. Avatar graphics range from human-looking characters with jointed limbs, to animated cartoon characters, to static pictures. Some graphical worlds allow users to choose among several pre-made avatars. Others allow you to make your own avatar by providing an image, choosing several body parts to put on an avatar body, or sticking on various props such as hats and eyeglasses to pre-made avatars. In text-based worlds, players customize their characters by writing text descriptions describing their real life selves, or more frequently, an imaginary online persona.

These worlds are generally divided up into rooms, with doors or exits that lead from room to room. In some 3-dimensional worlds, rooms are much larger, more continuous spaces. Instead of dividing rooms with hard boundaries, these worlds limit what your character can see and hear by drawing a circle of a given radius around your avatar's current location. Graphical rooms contain some sort of background image, and perhaps images of other objects in a room. In text-based worlds, rooms are described with words. The room description is displayed whenever a user walks into a new room, as well as the names of the other players and objects that are currently in the room.

Users typically communicate with each other by typing messages, which are seen by anyone in the room. Some worlds also have an audio system where users speak into a microphone and can be heard by others in the room. Most have mechanisms where players can send messages to specific individuals who are currently in other rooms, or where players can whisper a message to just one person at a time.

1.6 Text-Based Online Worlds

Text-based online worlds have been around since the late 1970s. These worlds are commonly termed Multi-User Dungeons, or MUDs, after their earliest incarnations as adventure games where players wandered around virtual dungeons collecting treasure and killing monsters. While many adventure-style MUDs still exist, there are now many other types of MUDs as well. Role-playing MUDs, where players act out parts in a favorite book or assume roles in solving a mystery are common. Special purpose MUDs for kids, businesses, or professional organizations flourish as well. Many current-day MUDs allow their members to help extend the virtual world, by making new rooms, creatures, and other virtual objects. Many of these MUDs, including LambdaMOO, MicroMUSE, and MOOSE Crossing, have extensive programming languages built into the virtual world that help users program their creations to give them behaviors.

1.7 Graphical Online Worlds

With the advent of fast, cheap computers that can support high-quality graphics, many companies and research labs have recently begun working on graphical MUDs, sometimes called graphical Multi-User Virtual Environments (MUVEs). They include multi-user game worlds such as Ultima Online and Terra. Others are virtual meeting places such as Microsoft's Comic Chat. Many are virtual worlds that users can explore, such as The Palace, WorldsAway, AlphaWorlds, and OZ Virtual. Intel Corporation's IDMOO 2.0, which was later converted into a Multi-User Software Developer's Kit (MU SDK), is yet another example of a graphical MUVE architecture. Many of these environments have carefully rendered 3d graphics, and some even have avatars that look and walk like people. However, only a very few provide any way for end-users to construct new things within the virtual world[1], and none makes it easy for novice users to script behavior in the world.[2] Many of these graphical MUDs boil down to embellished chat rooms inside very beautiful and carefully crafted graphical worlds.

[1] AlphaWorlds is the only graphical world that allows users to build things in the virtual world. Users can copy objects and claim land. With these tools, some users have been able to make complex virtual palaces for themselves. However, these tools are not accessible to kids and novice computer users.

[2] The Palace has a user-accessible programming language called Iptscrae, but it is primarily intended for system administrators to use when building new worlds and is admittedly "a little intimidating to look at" (Palace Support Online Manuals 1997). Some end-users have successfully written programs for their graphical avatars in Iptscrae, but this programming is limited to controlling your own avatar. The Palace does not allow end-users to construct new parts of the graphical world.

2 The Pet Park Project

2.1 Goals

The Pet Park project had many goals. Pet Park was to be a virtual world that kept the constructionist, community spirit of MOOSE Crossing, but used graphics as its primary medium instead of text. Unlike many commercially available graphical online worlds, it should allow users to build by creating rooms, creatures, and other objects. Pet Park should be accessible to young children, perhaps as young as six or seven, since it wouldn't need to rely on users being able to read or write in order to get around the virtual world. Yet Pet Park should have a high ceiling, allowing kids to continue to be expressive and develop their skills as they learn more. Pet Park should be compelling for a wide age range, allowing kids of many ages to learn, teach, and play together in a community learning environment.

From the onset, I knew that I wouldn't have enough time to fully build Pet Park to meet all of these goals, test it with kids, and write a thesis in one year. Hence, at the very beginning of the project I articulated my vision of Pet Park, what Pet Park would be if I had all the time and resources I needed to complete it. The actual Pet Park prototype was based very strongly on this initial vision, but omitted certain functionality and user interface features due to time constraints. The following section describes this initial vision of Pet Park.

2.2 The Vision of Pet Park

The vision of Pet Park is a kind of graphical MOOSE Crossing where kids can design and build things in a graphical virtual world. Like MOOSE Crossing, kids log into Pet Park over the Internet to talk with other kids and help build the virtual world.

They form a learning community, designing and building the virtual world together, and helping each other with their projects. Instead of being entirely text-based like MOOSE Crossing, however, Pet Park is a 2-dimensional graphical world. This difference makes Pet Park a somewhat different kind of learning environment from MOOSE Crossing. No longer is there a large component of creative writing and reading; instead the focus of Pet Park is on building the virtual world by programming animated behaviors for creatures and other objects in the world.

Kids can create new objects to populate the world, such as a dog that waves whenever its owner enters the room or a sofa that says "Ouch!" whenever anyone sits on it. Kids can create new places or "rooms" for the virtual world, such as a pet store where you can buy a dog, or a game room where two kids can play checkers. Kids teach their creatures behaviors with an object-oriented programming language designed for kids, called YoYo. Through these kinds of design and creation experiences, kids will get excited about their programming projects and will show off their programs to other kids. In the process, they will learn important ideas about computer programming, including procedural abstraction, using variables, and debugging.

The Pet Park world starts out with a few interconnected rooms and a large library of example graphical objects which kids can use as models for things they can build themselves. Each object in the library comes with a given graphical appearance, as well as a large set of "building blocks," or small animations, of that creature. Every object in the world has a set of properties, persistently stored variables that keep track of an object's name, location, coordinates, etc., as well as a set of scripts that define the building blocks and other behaviors for that object. Pet Park is an object-oriented environment, meaning that kids make new creatures by creating children of existing objects. The child of an object inherits all of the scripts and properties of the parent object.

For example, one of the dogs in the library might be a brown cocker spaniel with floppy ears named Rory. Rory comes with animated building blocks like "wagTail," "walk" "jump," and "laugh." A kid could make a child of Rory in order to make her own pet dog, Grover. She could then use Rory's building blocks in Grover's programs, since Grover inherits all of his parents' building block behaviors. For instance, she might teach Grover a simple dance with a script like this:

```
wagTail
walk
turnAround
walk
repeat 3 [jump]
wagTail
```

In this program, Grover will perform the animations for the building blocks called "wagTail," "walk," "turnAround," and "jump" in the order given, forming a little dance.

She might later make an improvement on Grover's script, using a property to store whether or not Grover is in a dancing mood, and changing her dance script appropriately. Now Grover's dance script might look like this:

```
ifelse (wantToDance = "yes")
[
   wagTail
   walk
   turnAround
   walk
   repeat 3 [jump]
   wagTail
]
[
   say "Sorry, I'm not in a dancing mood right now."
]
```

Building blocks might also include sounds along with the animation. For example, a dog's "bark" building block might include both an animation of a dog picking up its head and barking, as well as the sound of a dog's bark. Some of the building blocks are very short animations consisting of just a few frames, such as "turnLeft" or "lookUp." Others could be much longer animations, such as "cry" or "jump." The building blocks are intended to be conceptually easy-to-handle blocks that seem natural to build with, and are not necessarily the same length or complexity.

Each building block is, in turn, a program consisting of the sequence of pictures that are played to make that animation. Advanced users can write programs at the building block level, substituting new pictures into a particular building block, or reordering existing picture frames to define a new building block. In this way, advanced users could even create an entirely new object, with a brand new graphical representation.

Similarly, kids can create new rooms in Pet Park. Rooms are simply a special case of a graphical object like a dog or other creature. Rooms have a graphical part, the background picture for that room. However, to prevent rooms from using inordinate amounts of graphical processing resources needed to display a changing background image, rooms don't have animation building blocks like other graphical objects. This means that rooms can't run programs that would change the background picture. Instead, kids can use objects to decorate the room. For instance, suppose a kid wants to make a room that looks like a forest with a waterfall in it. He could start with a simple background that has a ground and sky painted on it. He could then add a few trees by making copies of tree objects from the library. Finally, he could add a waterfall object, and use the waterfall's building blocks to write a program that makes the water run.

2.3 The Pet Park Prototype

The Pet Park prototype follows the vision of Pet Park quite closely. The only major difference is that the prototype does not allow kids to incorporate their own drawings into the virtual world. Instead, kids make use of a library of objects from which their own creations may inherit their graphics. Additionally, building blocks do not currently include sound effects, such as the sound of a dog barking with the "bark" building block.

This section describes the design of the Pet Park prototype, including the underlying software architecture, the client user interface, and the design of the virtual world itself. It concludes by describing the afterschool program where a group of four kids used Pet Park for two months, and shows a sampling of the kids' projects.

2.3.1 Software Architecture

The foundation of the Pet Park virtual world is formed by integrating two existing components: Intel's Multi-User Software Developer's Kit (MU SDK)[3] and the YoYo programming language interpreter[4]. The MU SDK is a software architecture implemented in Java that takes care of most of the basics involved in having a graphical, multi-user space, but does not support easy end-user programming of objects in the world.[5] The MU SDK takes care of basic tasks such as displaying graphics, setting up a multi-user session, connecting clients, and maintaining objects.

The MU SDK forms the base architecture for Pet Park. Pet Park then incorporates the YoYo programming language interpreter, also written in Java. The

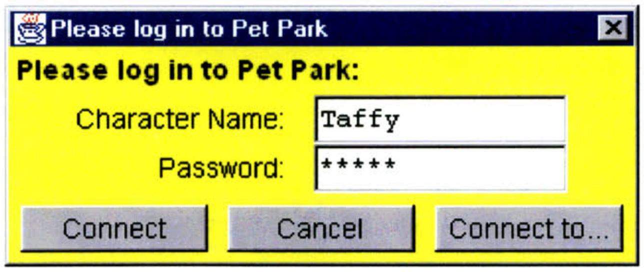
[3] The MU SDK is the property of Intel Corporation.
[4] The YoYo interpreter was originally developed under the name "Cocoa," and later, "Bongo," by Andrew Begel of the Epistemology and Learning Group of the MIT Media Lab (Begel 1997).
[5] Advanced users **can** write Java programs for their objects in the MU SDK. This programming is done offline, and requires a detailed knowledge of the MU SDK system architecture.

YoYo interpreter implements a kid-friendly programming language layer that sits on top of Java. With YoYo in place, kids can write YoYo programs for their objects in the virtual world. Kids can execute their YoYo programs, which can, in turn, change the graphical state of the corresponding MU SDK object. In this way, kids can write programs that animate their creatures and give them behaviors. Much of the implementation work in developing Pet Park was in integrating YoYo into the MU SDK framework.

Pet Park also required two new elements in addition to this basic architecture. The first addition was an account registration system, so that kids could have a username and password and would be able to safely store their programs for later use. The second was a modification to the MU SDK server so that objects don't disappear from the shared virtual world when their owners disconnect from Pet Park. It is important for kids' projects to be persistent, both to act as example projects for other kids, and to fit with the model of a community building space. Pet Park wouldn't seem like much of a building space if it were virtually empty most of the time.

2.3.2 Client User-Interface

This section will take a tour of Pet Park's client user-interface. The client interface was designed especially for Pet Park, though it borrows some interface ideas from MacMOOSE and JavaMOOSE, the clients developed for MOOSE Crossing.



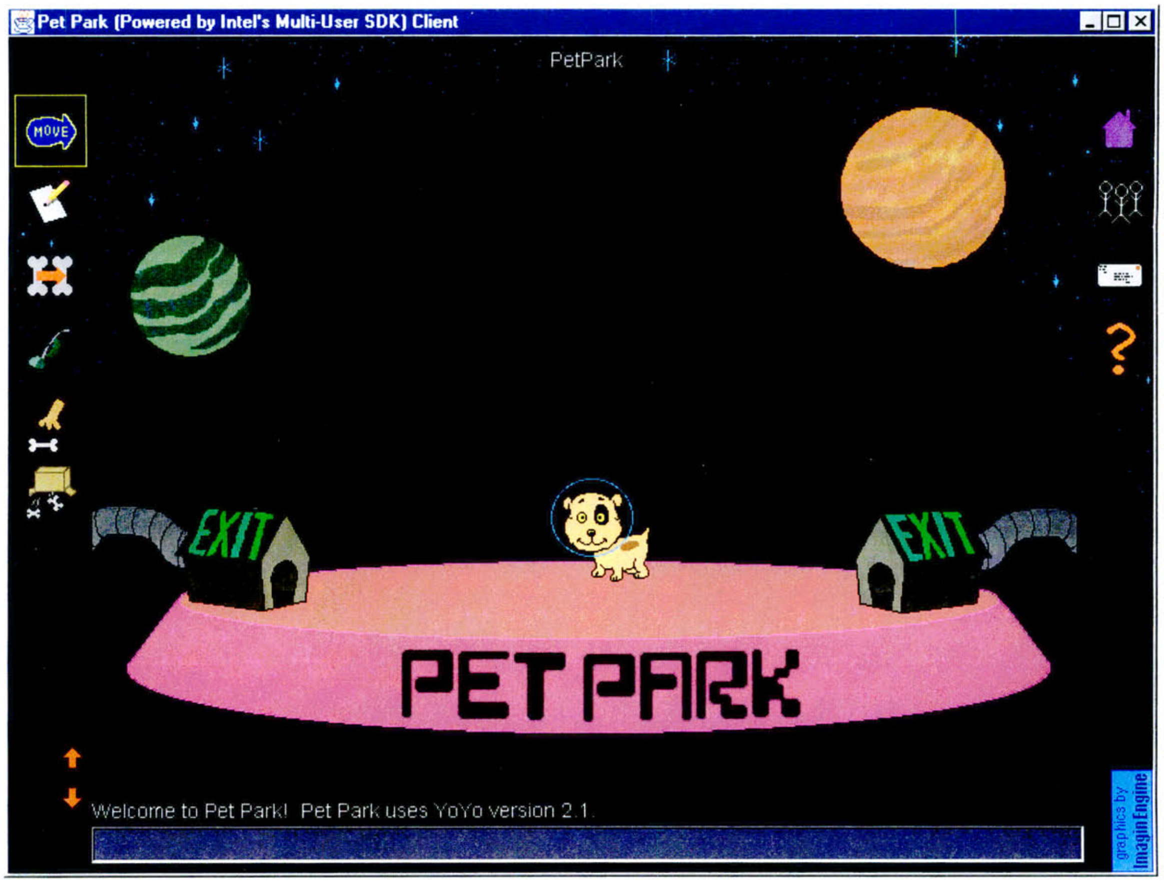
Please log in to Pet Park:

Character Name: Taffy

Password: *****

Figure 1. The login dialog.

When users first start up Pet Park, they are asked to login with their character name and password, as in Figure 1.



Welcome to Pet Park! Pet Park uses YoYo version 2.1.

Figure 2. The main window.

After a successful login, the user's character appears in the main window, as in Figure 2. The main window of the Pet Park client has several parts. The graphical display of the current Pet Park room is located in the center of the main window, taking up most of the window space. The display of the current room consists of a background image and several graphical objects, such as the planets, doghouses, and the dog character. These graphical objects can be examined and manipulated with various tools in the user interface, located around the perimeter of the room view. Each of these control and output areas will be explained in turn.

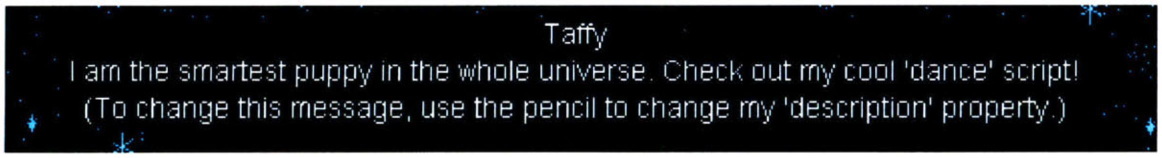

Taffy
I am the smartest puppy in the whole universe. Check out my cool 'dance' script!
(To change this message, use the pencil to change my 'description' property.)

Figure 3. The status pane.

Top: The Status Pane

The status pane at the top of the window displays multi-line text descriptions and other quick help pointers. As the mouse is moved over a character or other object in a room, a message appears in the top status pane with its description, or other information about how to use or edit that object. Figure 3 shows the message in the status pane when the mouse is moved over Taffy, the dog in the main window in Figure 2.

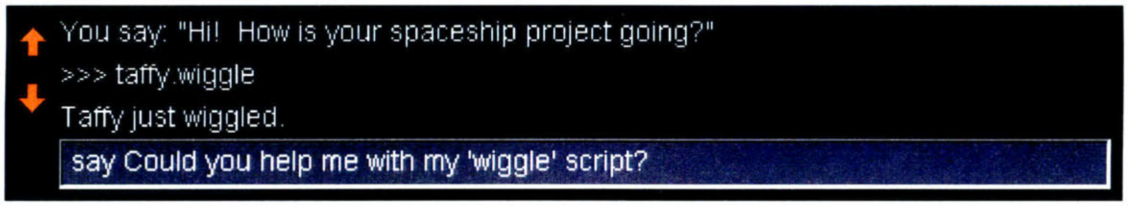

```
You say: "Hi!  How is your spaceship project going?"
>>> taffy.wiggle
Taffy just wiggled.
say Could you help me with my 'wiggle' script?
```

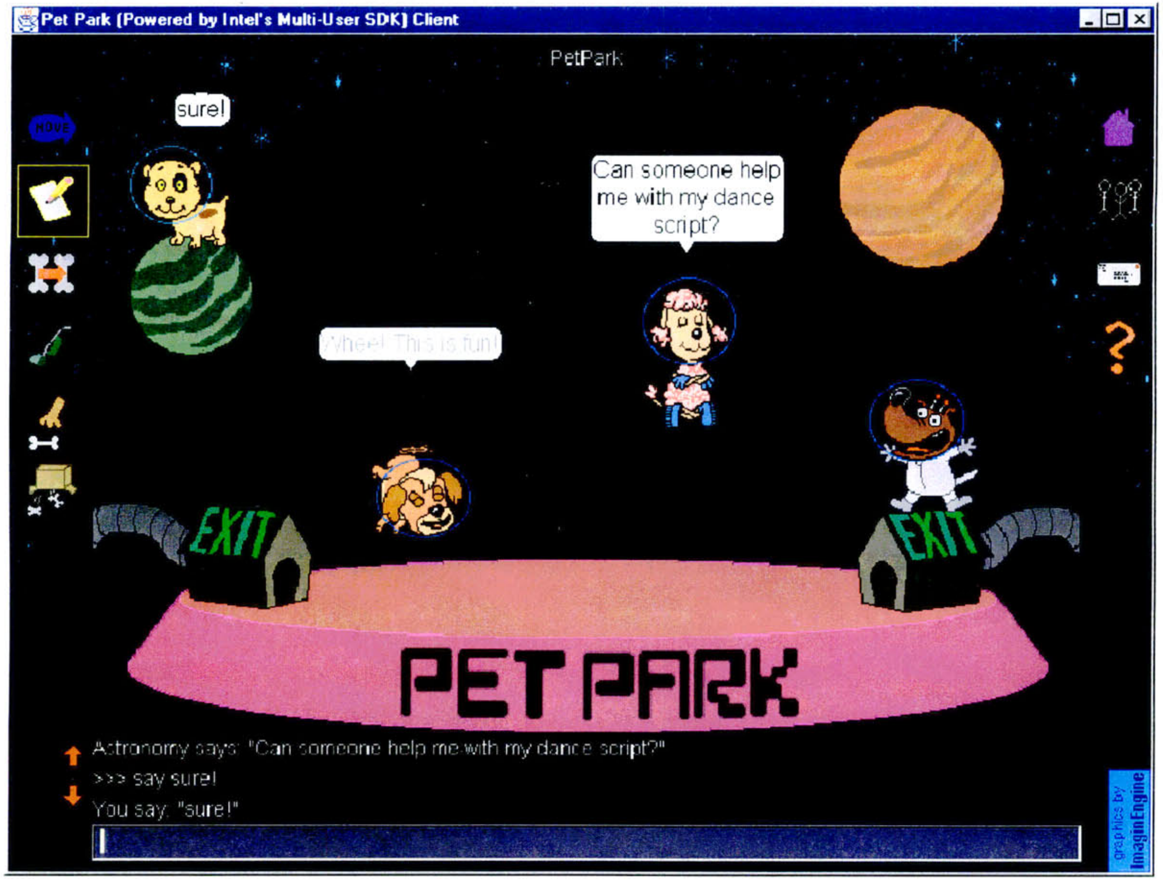
Figure 4. The input and output boxes.

Bottom: The Input and Output Boxes

An input box at the bottom of the window is one way for the user to interact with the system, shown in Figure 4. Above the input box is a buffer that caches the most recent 100 commands typed in the text input box, along with any printed responses. Two red arrows let kids scroll through this buffer to see what has been said recently, or to recall a command typed earlier in their Pet Park session. Kids primarily use the input box to talk to other characters in the current room by typing "say" and then the message they want to display. For instance, a new Pet Park member might type:

```
say Hi!  I'm new here, can you help me?
```

When a child says something in Pet Park, two things happen. First, a cartoon speech bubble containing the message appears over the child's character in the room view. The text in the speech bubble fades after a few seconds, lending a visual guide to the time-


Pet Park (Powered by Intel's Multi-User SDK) Client
PetPark
MOVE
sure!
Can someone help me with my dance script?
Wheee! This is fun!
EXIT
EXIT
PET PARK
graphics by ImaginEngine

```
Astronomy says: "Can someone help me with my dance script?"
>>> say sure!
You say: "sure!"
```

Figure 5. Several characters talking with one another.

order of the conversation.[6] Second, a text transcript of what was just said is saved in the output text box. See Figure 5 for an example of several characters talking with one another.

The input box can also be used as a command line to the YoYo programming language interpreter. More advanced users can run their objects' scripts directly by typing a YoYo command such as

```
taffy.dance
```

[6] This is a built-in feature of the MU SDK.

which would run Taffy's "dance" script. Users can also check on the values of their object's properties from the input box, or try out commands that they might want to later use in their scripts.

Left: Mouse Tools

Two columns of tools are stacked at the left and right edges of the room view. The tools in the left-hand column define the behavior of the mouse pointer in the room view. They are intended to work like tools in a computer drawing program. Users can click on a tool to pick it up, then click on an object in the main interaction window to use the tool on that object. Only one tool may be used at a time. The current tool is highlighted in yellow.

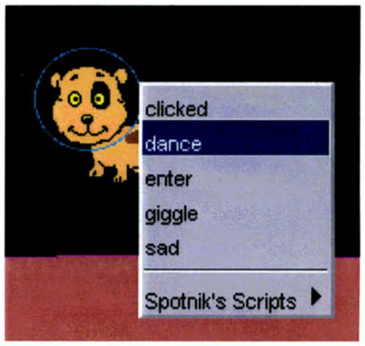


Figure 6. Clicking on Taffy with the arrow tool.

The default tool is the *arrow*, which allows the user to view and execute an object's scripts. Clicking on an object with the arrow tool brings up a popup menu listing that object's scripts. Users can pick among the choices in the popup menu to execute that script on their object. Figure 6 shows the effect of clicking on Taffy with the arrow tool.

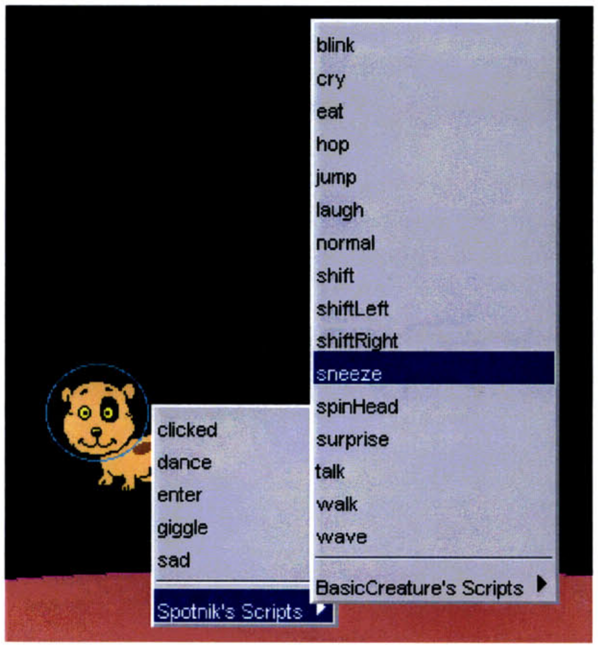
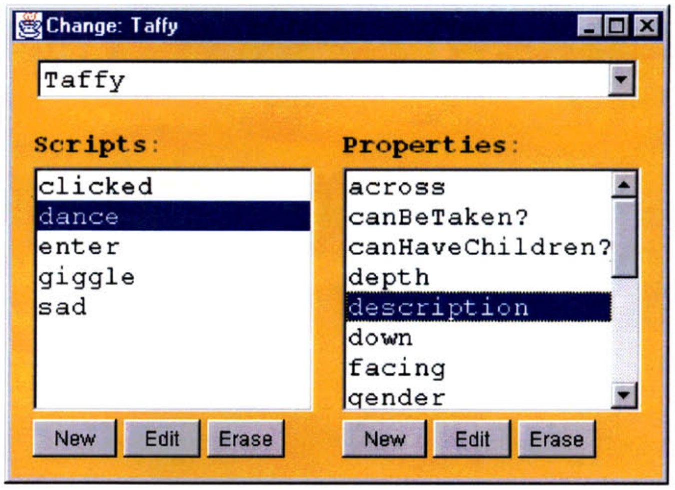


Figure 7. Climbing the inheritance hierarchy.

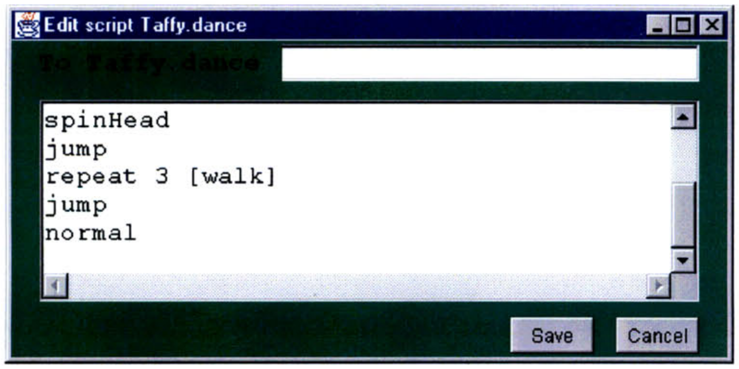
Taffy has five scripts: clicked, dance, enter, giggle, and sad. Users can climb up the inheritance hierarchy within the popup menus to get access to an object's parent's scripts as well. Figure 7 shows the scripts for Spotnik, Taffy's parent. Spotnik has sixteen scripts defined on it, beginning with "blink" and ending with "wave." The user could go on to view and execute the scripts for all of Taffy's parents through the cascading menus.



```
Scripts:        Properties:
clicked         across
dance           canBeTaken?
enter           canHaveChildren?
giggle          depth
sad             description
                down
                facing
                gender
```

Figure 8. The object browser window.

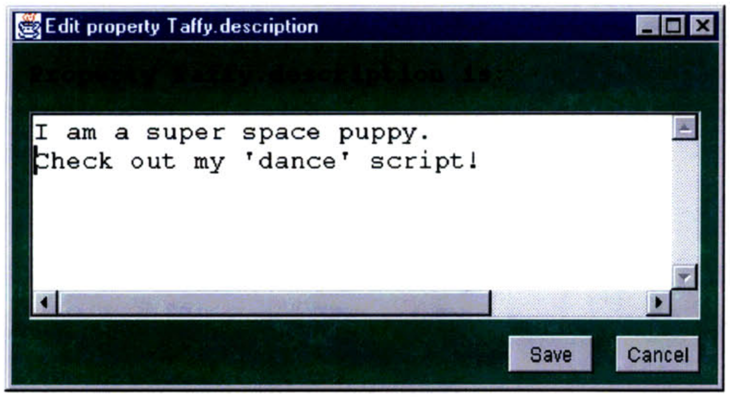
The *edit* tool allows users to view or modify the scripts and properties of an object. Clicking on an object with the pencil opens a new window that lists that object's scripts and properties, as in Figure 8. The *new* button makes a new script or property. The *erase* button erases the highlighted script or property, after passing through a confirmation dialog. Users double click a script or property name to open up an editor[7] for that item, or highlight the item and click the *edit* button. Only the owner of an object has permission to make changes to that object's



```
spinHead
jump
repeat 3 [walk]
jump
normal
```

Figure 9. The script editor.

[7] Andrew Cheng is working on an alternative editor for writing YoYo programs in Pet Park that works by linking together graphical puzzle pieces instead of a writing a program with text commands. This is a separate Masters thesis project.



Property Taffy.description is:

```
I am a super space puppy.
Check out my 'dance' script!
```

Figure 10. The property editor.

scripts and properties, but any user can view them. Figure 9 shows the script editor for Taffy's "dance" program. Figure 10 shows the property editor for Taffy's "description" property.

The *make a child* tool is used to make a child of an object in the virtual world. Players can make a new creature by clicking on an object with the make a child tool. They are prompted to enter a name for the object. As long as the name is unique, the new object is created and appears in the main room window next to its parent.

The *erase* tool is used to permanently delete an object. Players can erase their objects by clicking on them with the erase tool. A dialog box confirms that the user really wants to erase that object. Users may only erase their own objects.

The *take* tool is used to take objects. Clicking on an object with the take tool allows a character to hold that object, giving the user control over moving it around the screen and moving it between rooms. When a character is holding an object, that object automatically follows the character as the character moves to different rooms.

The last tool on the left-hand side is the *drop* tool, which performs the inverse function of the take tool. Clicking on an object with the drop tool will leave that object in the current room at its current coordinates.

Right: Action Tools

The four action tools in the right-hand column are one-time actions, and have no effect on the state of the left-hand tools.

The *home* action transports the player to the player's house, or to the center of Pet Park if the player hasn't made a house yet. Players can change their "home" property to define where this button will take them. The home button helps players reorient themselves when they get lost exploring the large Pet Park world. It also can be used as a hotkey to get to a favorite location, frequently a player's home room or the library.

The *who* action brings up a window that lists the users currently logged in to Pet Park, and their current locations in the virtual world.[8] Double-clicking on a player's name in the who list moves your character to that player's current room location.

The *mail* action brings up a window that acts as a small, in-world e-mail client.[9] Users can receive email, send email, and subscribe to email discussion lists. Users can also create new discussion lists on topics of interest.

The *help* action brings up a window that can be used to search for help on various aspects of Pet Park, including talking to other players, writing scripts, and sending mail.

[8] The *who* action was not implemented for the Pet Park prototype.
[9] The *mail* action was not implemented for the Pet Park prototype.

2.3.3 Designing the Virtual World

The previous section described the user interface components of the Pet Park client. Another important part of Pet Park is the design of the virtual world itself. The most visible part of Pet Park is the graphic theme: a park for pets in outer space. The theme was chosen in collaboration with the artist who designed the graphics for Pet Park.[10] We wanted to pick a theme that would appeal to both boys and girls from a wide age range, from seven-year-olds to thirteen-year-olds and beyond. It was important for the theme to provide some amount of structure for kids who need help generating ideas for projects, yet it needed be open-ended enough that it wouldn't put too many limits on kids' project ideas. Nearby planets in the space world would allow kids to define different neighborhoods, each with potentially different flavors of construction and environment. An outer space world without gravity also solved the aesthetic problem of being able to have the characters float around above the ground without falling.

The graphic style was also important. We chose a cartoon style, with black outlines, and solid colors. Looking forward to having kids use their own drawings in addition to the pre-built characters, it was important to make it easy for kids to be able to modify and mimic the graphic style of the objects. Having shaded, 3-dimensional images as the defaults would make it much harder and less appealing for kids to make their own graphics, since their own creations would pale in comparison with the default, professionally drawn images. The graphic artist made it easy for kids to change the colors of an object by connecting regions of similar color, making it easy to repaint a green chair into a blue one in one step.

[10] Mark Loughridge and his colleagues at ImaginEngine Inc. designed the graphics for Pet Park.

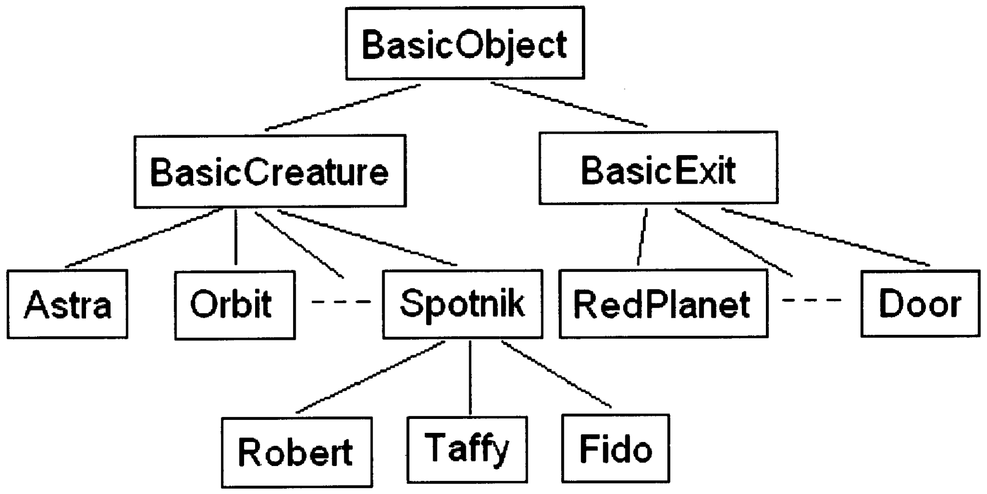


Figure 11. The top of the object hierarchy.

Pet Park is an object-oriented construction environment. Since YoYo is an object-oriented programming language, all of the objects in Pet Park, including kids' characters, exits that lead to other rooms, props, furniture, and even the rooms themselves are objects in a centralized object hierarchy. The object hierarchy is sketched in Figure 11. Having an object hierarchy helps kids get started quickly when they first get their Pet Park account. They start off with a dog that has already inherited animated building blocks that can be played with right away, instead of having to build behaviors for the dog from scratch. For instance, Taffy is a new user on Pet Park, but since her character inherits from Spotnik, Taffy's character already knows many animated behaviors. The object-oriented system also helps kids build on each others' work. For example, a girl could make a dog that tells jokes, then a boy could build on her work by making a child of that dog, and teaching it how to chase its tail in addition to telling jokes.

Scripts on creatures, exits, and other objects can be executed several different ways. One is by calling the script directly at the command line input box. Another is by selecting the script from an object's popup menu. Some special-purpose scripts can be executed in other ways as well. For instance, double-clicking on an object will run its script named "clicked," if it has one by that name. "Enter" scripts get executed whenever that object enters a new room, or whenever someone enters the room that object is currently in. Similarly, "bump" scripts get executed whenever objects bump into each other.

The YoYo programming language is an interpreted language, rather than a compiled one. This means that both syntactic and conceptual errors in kids' scripts will only present themselves when that script is executed, not when it is saved. For comparison, the MOOSE Crossing programming language, MOOSE, was a compiled language. This meant that most errors that kids would make in their scripts would be flagged with an error message when kids save their scripts, rather than later on when they execute the script to test it out. Some errors that were not caught by the compiler would still show up during execution, however, leading to two different kinds of script errors: ones that appear at compile-time, and ones that appear at run time.

There are obvious pluses and minuses to these two approaches, and neither is clearly superior to the other. In the interpreted language, it's nice that all errors are flagged at the same stage in the process, at run time. However, this stage is later in the program development process that one would like, when the editor containing the script in question has probably already been closed. Fixing the bug requires repeatedly switching between the script editor and the main window, which can be annoying if the user interface does not support this well. On the other hand, compiled languages catch most common errors while the user is still modifying the script in the script editor, which allows the user to fix the bug earlier in the development process. However, getting through the script editor without errors doesn't mean that the script is bug-free. The harder-to-fix logic errors show up at run time, leading to the same problems as in the interpreted case.

A crucial part of the graphic design was the design of the Pet Park characters, the graphical avatars kids would use to represent themselves in the world. Figure 12 shows the five pre-built avatars the kids can choose among, each showing a frame from one of their behaviors. These avatars were designed to appeal to both boys and girls, and to be fun characters to play around with. Each of the characters comes with a set of nine core building blocks: cry, eat, laugh, normal, sneeze, surprise, talk, walk, and wave. In addition, each character has between four and ten other building block behaviors, including ones such as spinHead, float, dance, jump, wagTail, and wiggle. In total, each character comes with more than three hundred frames of pre-drawn animations. The avatar characters are children of the BasicCreature object in the hierarchy, which defines scripts such as turnLeft and turnRight that allow the avatars to face in either direction.

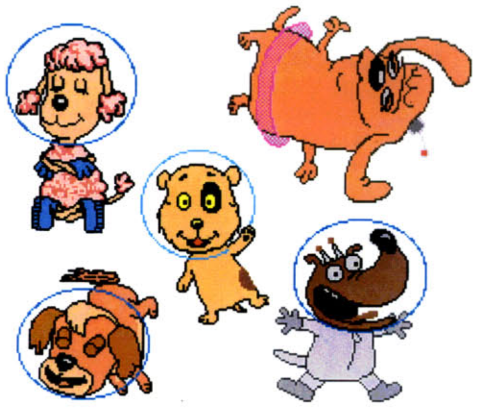


Figure 12. The five characters. Clockwise from top left: Astra, Twinkle, Orbit, Star. Center: Spotnik.

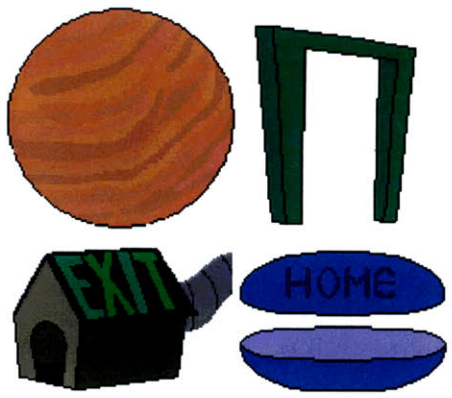


Figure 13. Some exit graphics.

Users move their objects within a room by dragging them with the mouse, enabling kids to place lamps on top of tables, or have their characters perch on a planet or sit on a doghouse. Kids move their characters between rooms in the virtual world by double clicking on a door, a doghouse, or a planet. These linking objects, or exits, have a "destination" property that determines where they lead, and are children of the BasicExit object in the main object hierarchy. Figure 13 shows a sampling of the graphics used as exits in the Pet Park virtual world.

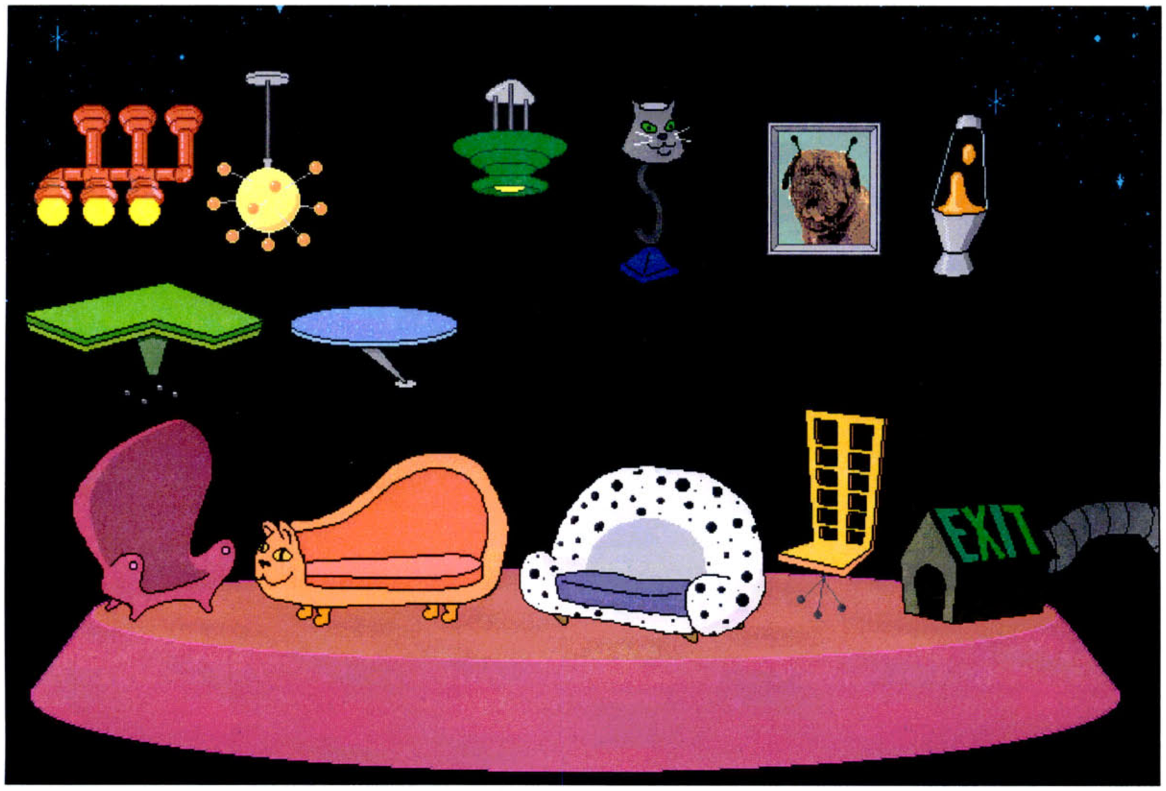


Figure 14. The furniture library.

Another important component of the Pet Park world is the library, which contains sample objects that kids can use as parents for their own creations. The library in the Pet Park prototype is divided into four sections: the Pet Library, the Furniture Library, the Thing Library, and the Exit Library. Each library contains many sample objects in its domain. Some sample objects come with a few scripts, others are primarily just graphics images that can be used either a static objects, or as a framework for kids' own scripts. Figure 14 displays the Furniture Library.

Pet Park is an open virtual community, but does implement some rules of ownership and privacy. There are no locked doors in Pet Park; kids can visit each others' rooms at will, though are asked not to bother other kids if they are asked to leave. Kids are expected not to leave things in public rooms, but rather should keep their objects in their personal home rooms when they log out. Anyone can view the scripts and properties of any object in the world, regardless of who owns it. However, only the owner can modify the scripts or properties of that object. The owner can set properties on her object that determine whether other kids can take the object, whether that object can have children, and whether other kids can run that object's scripts.

Like MOOSE Crossing, Pet Park uses the notion of a quota of objects, or an object conservation system. In order to keep the number of objects per client reasonable for computational purposes, and to encourage kids to work on a few objects, rather than trying to copy the entire library, kids are restricted in how many objects they may own. Just like responsible ecology-minded citizens conserve water to ensure that there is enough for everyone, Pet Park members are required to conserve on the number of objects they make. By default, players are allowed to create ten new objects, not including their own character object. A system administrator may increase a particular child's quota as necessary.

2.3.4 The Kids and their Projects

As part of the evaluation phase of this project, I held a weekly afterschool program for two months with a group of four kids between the ages of 8 and 13. Table 1 lists the kids'

Character Name	Sex	Age
Leah	Girl	8
Chris	Boy	9
Bob	Boy	10
Rebecca	Girl	13

Table 1. Kids in the afterschool program.

names and ages.[11] The kids came to the Media Lab for a two hour session on Wednesday afternoons, though some of the kids often stayed for as long as three or four hours as their parents indulged them. During the sessions, I helped the kids with their projects and took notes on their progress and on aspects of the environment with which they had difficulty. The goal of the afterschool program was to do some limited testing with kids in order to help assess the design of Pet Park, not to do a formal study of their learning experiences. There was simply not enough time to undertake a more formal study of this nature, though it certainly would have been interesting to do. In addition to the afterschool kids, 16 kids and 27 adults have characters they can use to connect to Pet Park over the Internet (as of May 20, 1998). This section will discuss informally my experiences working with kids using Pet Park, and will present some of the kids' projects.

The first session of the afterschool program presented a few very pleasant surprises. Once the kids logged in for the first time, I showed each of them how to talk to other players, how to move around between rooms, and how to run their character's building block behaviors. At first, the kids spent most of their time wandering around the virtual world, exploring around and getting their bearings, and played around with their character's building blocks. In short order Chris found the library, and discovered how to make new things. I suggested that he could claim one of the doghouses, and decorate it with things he could create from the library. The rest of the kids caught on quickly from Chris's lead, and all began making their rooms. Although technical problems with the object conservation system prevented them from making more than a few objects apiece instead of the usual ten, they enjoyed staking a claim on a piece of the virtual world.

[11] The kids' character names have been changed to protect their identities.

Once the kids couldn't make any more things for their rooms, I logged in and showed them the scripts on my character. Using my character's "dance" script as a model,

```
turnLeft
shiftRight
spinHead
jump
repeat 3 [walk]
jump
normal
```

each of them began to write their own scripts. By the end of the session, each of the kids had written between two and seven scripts for their own characters. Each of the kids had mastered the use of their characters' building blocks, and experimented with the "repeat" command. Here is a sampling of the kids' scripts from this first session:

Leah (girl, age 8). Leah wrote 2 scripts during the first session.

"Leah"

```
sneeze
sneeze
walk
jump
hop
blink
blink
blink
cry
cry
sneeze
```

"rock"

```
repeat 10 [surprise]
repeat 6 [sneeze]
repeat 4 [blink]
repeat 2 [spinhead]
```

Chris (boy, age 9). Chris wrote 7 scripts during the first session.

"dancedemonstration"

```
kick
kick
laugh
kick
```

```
kick
sneeze
```

"clicked"[12]
```
lookleft
lookright
laugh
lookright
normal
kick
kick
kick
kick
lookleft
normal
laughpt
```

"basic345"
```
repeat 3 [basic basic2]
```

"basic2"
```
kick
kick
kick
kick
kick
kick
laugh
```

Bob (boy, age 10). Bob wrote seven scripts during the first session.[13]

"sayimgoingtocry"
```
say "im going to cry"
repeat 3 [cry normal]
say "I think I am better now."
```

"allergy"
```
repeat 2 [repeat 2 [sneeze tossHead sneeze tosshead]]
```

"clicked"
```
kick
wait 1
cry
wait 1
sneeze
wait 1
laugh
wait 2
```

[12] Recall that any script named "clicked" is a special script that can be executed by double clicking on that object in the room view, in addition to the usual ways of executing a script.
[13] Bob had some previous programming experience in Logo which gave him an initial head start over the other kids.

```
kick
normal
```

Rebecca (girl, age 13). Rebecca wrote 5 scripts during the first session.

```
"Dance1"
laugh
spinHead
sneeze
talk
laugh
cry
laugh

"excited"
repeat 5 [Shiftright]
repeat 10 [laugh]
repeat 5 [shiftleft]
repeat 10 [laugh]
```

The kids used Pet Park for just under three hours that first session, almost a full hour longer than scheduled. They were excited and intensely interested throughout the session. I had expected them to get distracted partway through—three hours is a long time to focus on one task for any kid. To my great surprise, they were working hard on their creations right up through the moment when their parents finally insisted that it was time to go home.

Overall, the kids exceeded my expectations on two counts. Drawing from my experiences working with kids on MOOSE Crossing, I was surprised and pleased at how quickly the kids were able to make their own scripts and how much they enjoyed doing so. Many kids on MOOSE Crossing created their first script during their second session online, and very, very few wrote seven scripts within their first three hours online. Also, kids using MOOSE Crossing did not typically remain glued to their computers for three hours at a time—their attention generally lapsed after an hour and a half, or perhaps two hours. It's important to note that these differences are not statistically significant, due to the extremely small sample size of kids I worked with both on MOOSE Crossing and on Pet Park. Even though they are informal, these results are encouraging.

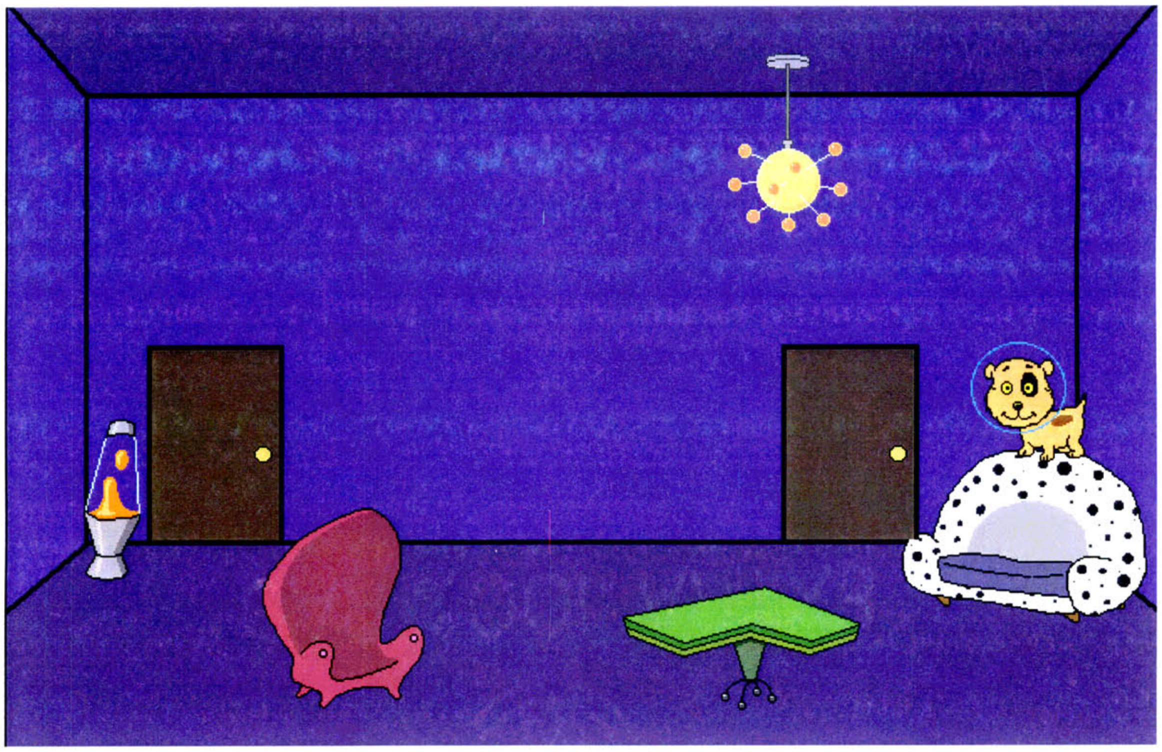


Figure 15. Rebecca's room (girl, age 13) after the second session. Rebecca's character is on the sofa.

The next two sessions of the Pet Park afterschool program continued to go very well. The kids wrote a few more scripts for their characters, though they spent most of their time making furniture and other objects in the library, and installing them in their home rooms. By the third session, the kids began to pay attention to the relative depths of their objects, carefully setting their objects' "depth" properties to ensure that lamps were drawn on top of tables, and that their character could sit on the couch, or hide behind it, as they wanted. Figure 15 shows Rebecca's room (girl, age 13) after the second session.

The kids also began exploring their characters' properties during these sessions, and some made text descriptions for themselves. Bob (boy, age 10) described himself as, "I am Bond , James Bond." and described one of his pets as "I am Bobs best friend. Ha Ha." Rebecca (girl, age 13) described her character as, "I'm a awesome pooch, and I live in the Purple Room." It was rare for the afterschool kids to give text descriptions to their objects; most objects bore their parent's default description and were never changed. It's unclear why the afterschool kids did not to describe many of their objects, especially since the handful of Internet users who have made things in Pet Park almost always gave their objects descriptions.

Bob (boy, age 10) began to write scripts for the objects in his room, having them greet players whenever they enter his room. A particularly clever, albeit simple, script of this genre caught my attention. Bob had noticed that all of the characters in Pet Park are dogs, and that his new lamp, which he had copied from the library, was the only cat. Naturally, then, the CatLamp should tease the dog characters whenever they enter the room. Bob gave his CatLamp a simple, one-line "enter" script:

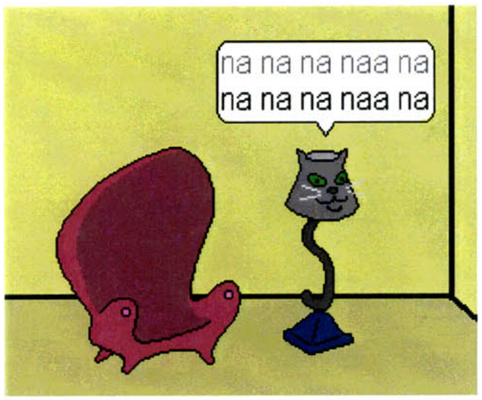


Figure 16. The Cat Lamp's "enter" script.

```
say "na na na naa na"
```

Figure 16 shows the CatLamp's response after someone enters the room.

Later sessions were plagued with technical problems, which kept the kids from progressing much further in their scripting and construction. The kids remained excited about their Pet Park sessions, but their progress was slowed tremendously because of frequent server restarts, and other distractions caused by technological failures. I would estimate that the kids were only able to work productively about one-third of the time during these later sessions, and with frequent breaks in their work. Although the kids were very patient throughout, they did occasionally comment that they forgot what they were working on and were going on to do something else, or decided not to re-create an object that had been lost in a crash.

Technical problems notwithstanding, over the final four weeks the kids continued to perfect their rooms, and two decided to move their belongings into new rooms on a planet I had recently added to Pet Park. Leah (girl, age 8) commented that she was excited about moving but that she was sad to be leaving her old house, and wondered if she should leave a sign telling others where she had moved.

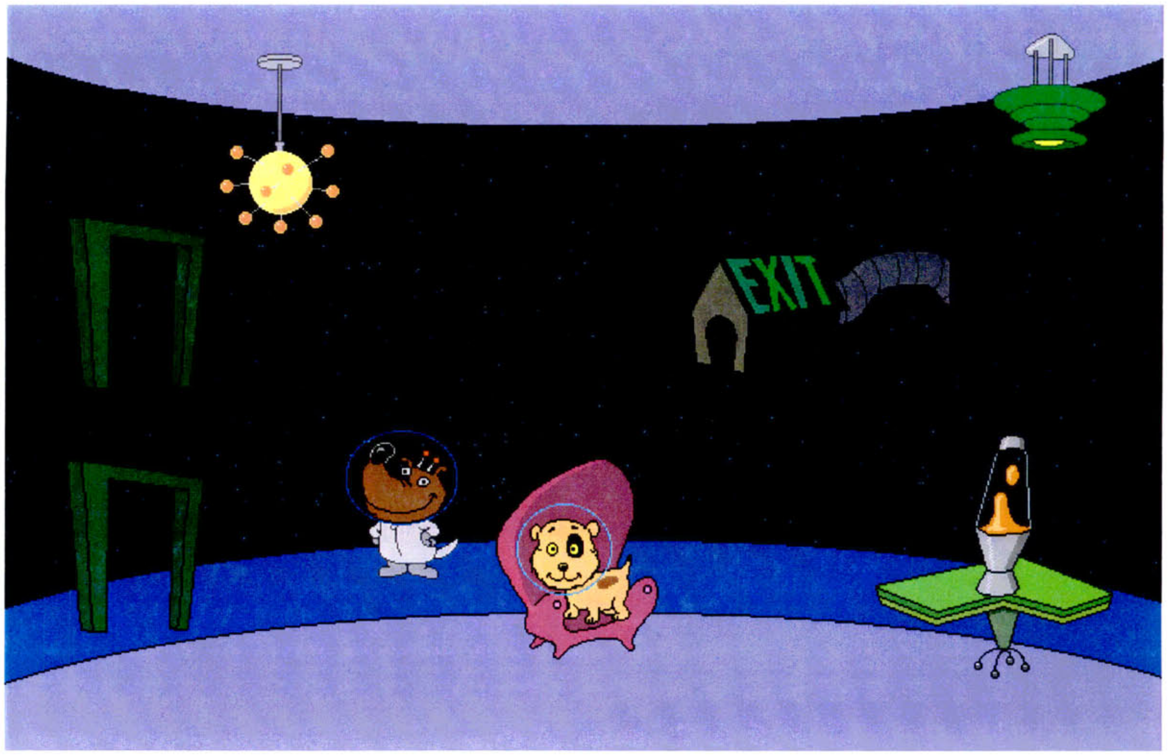
The kids continued writing slightly more sophisticated and longer scripts, though at a much slower rate. They also continued to make new objects from the library. Leah and Rebecca spent a lot of time arranging and re-arranging their rooms, and less time writing scripts. The boys both spent most of their time writing scripts, and comparably less time decorating their rooms. Although there were eight sessions, only two kids were present for all eight sessions: Bob and Leah. Rebecca missed three sessions, and Chris missed one. By the end of the program, Bob had written at least 21 scripts and Chris had written at least 22.[14] Leah wrote 5 scripts, and Rebecca wrote 12 scripts over the course of the program. The following is a sampling of the kids' later projects:

Leah (girl, age 8)

```
"hopp"
cry
cry
eat
blink
normal
```

[14] A few of the kids' scripts were lost in a server crash after the third session of the program. These numbers reflect the scripts that were not lost. Hence, these figures are a bit low, though not off by more than a few scripts per kid.

Leah only wrote one new script in the last half of the program, but she spent a lot of time designing her room. She spent one entire session moving from her old room in a doghouse to one of the new space pads. Here is her space pad house after the final session:



Chris (boy, age 9)

Chris wrote several new scripts for his character, like these:

```
"dance?"
say "This is avery ancient dance"
wait 1
look left
wait 1
look right
wait 1
danceC
kick
kick
kick
wait 1
look left
laugh
wait 1
```

```
say "That will just have been my performace!"
wait 1
laugh
wait 1
normal
```

"try"

```
surprise
say "AAAA!!"
wait 1
say "Ohhh sooory, I, er, thought you were a cat!"
wait 1
laugh
wait 1
normal
```

"eat-say"

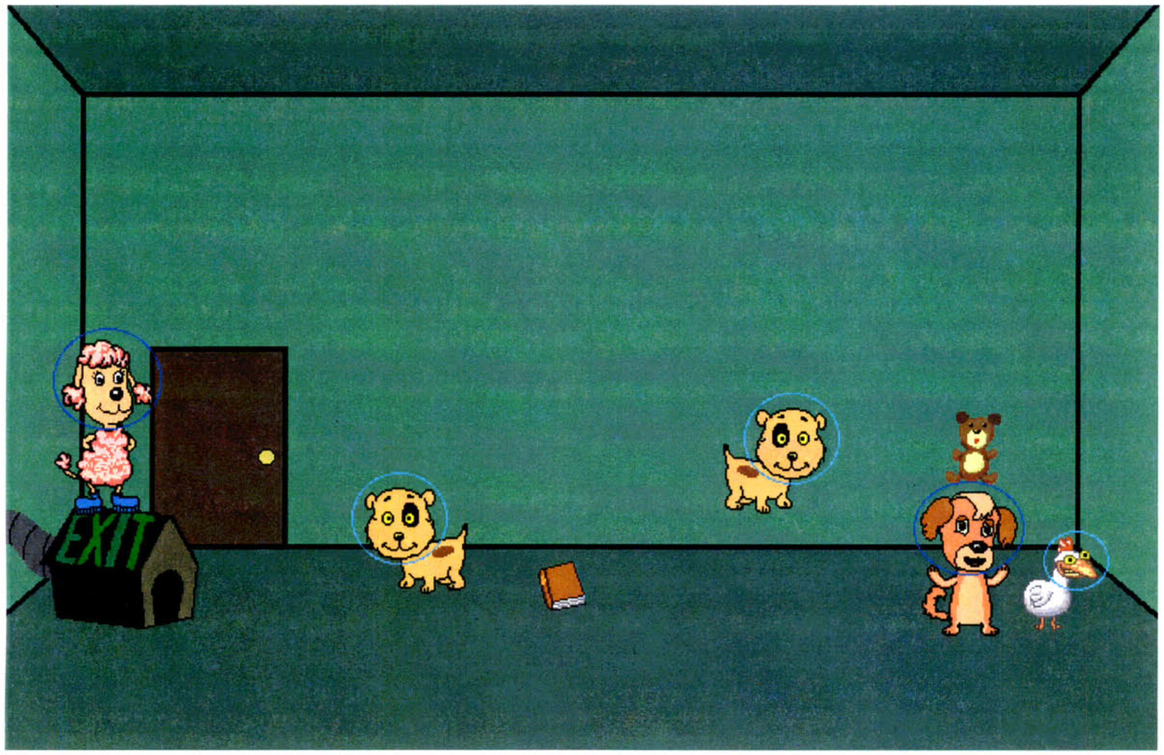
```
eat
wait 1
sneeze
wait 1
say "oops! I think I'm alergic!"
wait 1
sneeze
wait 1
laugh
wait 1
say "That was all fake!"
wait 1
sneeze
wait 1
say "realy, it was!"
wait 1
sneeze
wait 1
say "I,er,I admit, it wasn't fake,er,"
wait 1
sneeze
wait 1
say "I'd better leave because I think I'm alergic!"
```

Chris also made another pet named Spotodoggy, and wrote this program for it:

"shniltz"

```
say "hey, look at me!"
wait 1
say "I can do anything, look!"
wait 1
spin head
```

Here is Chris's room at the end of the final session:

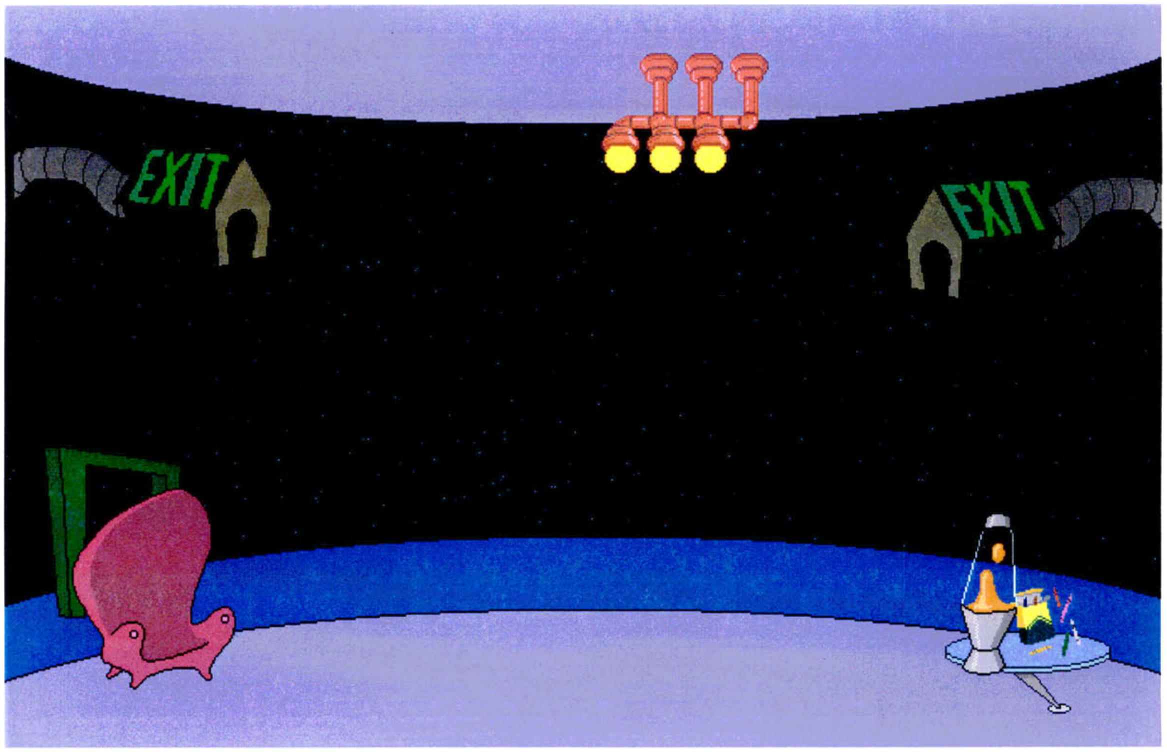


Bob (boy, age 10)

Bob spent most of his time writing scripts for this character, including ones like these:

```
"hungry2"
if [hungry = yes] [eat]
set hungry "no"
wait 320
set hungry "yes"

"clicked"
kick
wait 1
cry
wait 1
sneeze
wait 1
laugh
wait 2
kick
normal

"enter"
say "Your nice."
say "Your really nice."
say "Not me."
say "You."
say "Just kidding."
say "We are both really nice."
say "Esspesially me though."
say "Ha haa."
say "Sorry just kidding again."
say "We are both the same."
say "Hi Ho Fiddley Do."
say "Bye."
```

Here is Bob's room at the end of the last session:



Rebecca (girl, age 13)

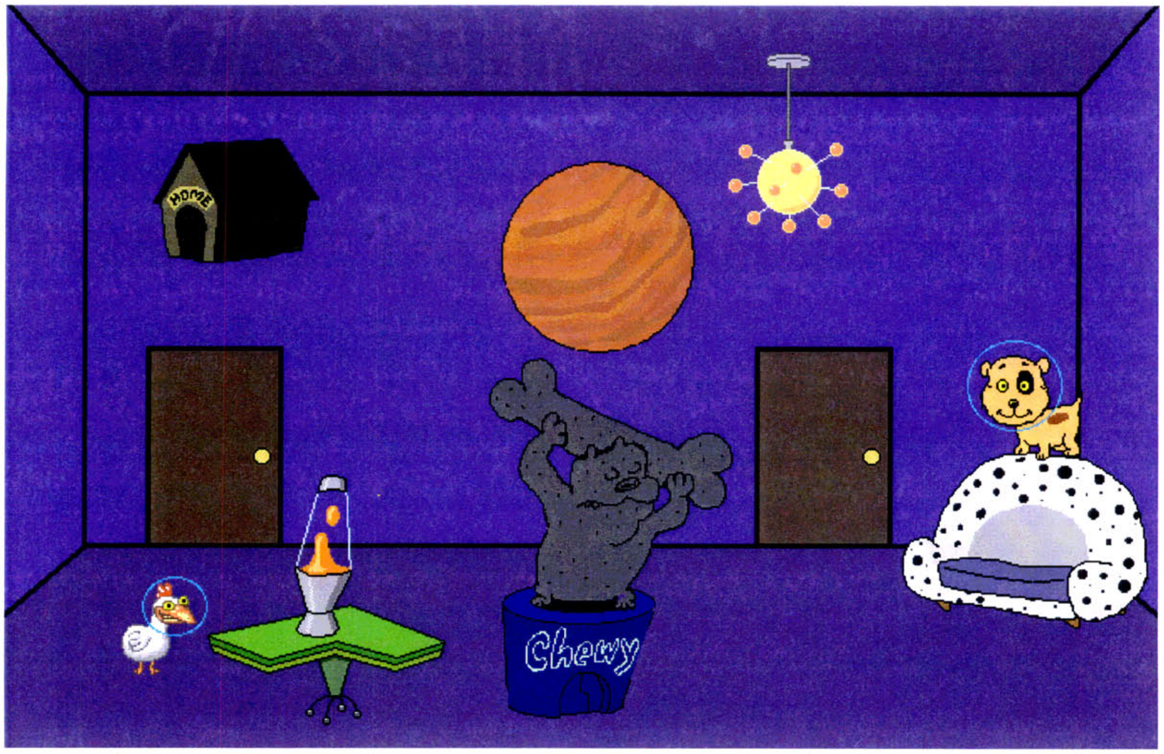
Rebecca made a chicken named crazycluck and wrote a few scripts for it:

```
"talk1"
say "Hello my name is crazycluck"

"talk2"
say "SQUAAAAWWK!!!!"

"talk3"
say "cluck cluck. I'm a chicken not a parrot!!!"
```

Rebecca also made a child of her new chicken, crazycluck, and named it "loonycluck." Loonycluck lives in one of the rooms on Planet Garon in the Pet Park world, greeting any passers-by.

```
"enter"
say "you are looking at an insane space chicken!!! HAHAHAHA!!"

"talk1"
say "Hi there I'm the first chicken on this planet"
```

Like Leah, Rebecca spent a lot of time online working on her room. Here is Rebecca's room at the end of the final session:



3 Graphics Change Everything: Exploring the Differences between Text and Graphics

When I began work on Pet Park, I quickly realized that simply adding a graphics layer to MOOSE Crossing was neither as easy as it sounded, nor was it necessarily the best approach for designing a graphical, constructionist world. Graphics are a fundamentally different medium than text, which had deep implications in what kind of environment Pet Park was going to become and what kind of learning experiences it would support. This chapter explores some of the key differences between text and graphics as primary media for constructionist, online worlds, using MOOSE Crossing and Pet Park as illustrations.

Although the following discussion limits itself to 2-dimensional graphical and text-based environments where kids and other novice computer users can build onto a virtual world, many of the differences presented here are equally valid for virtual worlds that don't encourage construction or aren't specifically targeted at novices. Furthermore, there are many hybrid online environments that combine text and 2-dimensional or 3-dimensional graphics in novel ways. Although these hybrids are not directly addressed by this discussion, they are certainly affected by these issues as well.

3.1 Affordances of the Medium

In text-based worlds like MOOSE Crossing, users form their own mental pictures of places and objects by reading descriptions and other text messages. For example, different kids might have tremendously varied mental pictures from reading this description of a MOOSE Crossing room:

```
The Highest Branch
You are on the highest branch of the tree that will support
any real weight.  It's a long way down!  There's a
beautiful view of all of MOOSE Crossing from up here--the
town to the east, the wilds to the west.  Clouds float
overhead.  You're so high up, you almost feel like you
could touch them.
```

By nature, descriptions leave an impression in the mind of the reader, but do not necessarily provide specific details of color, shape, size, or location. One child may see MOOSE Crossing from the Highest Branch as a lush, green park and a small cluster of houses, visible only through tangled tree branches. Another may see a clear, unobstructed view of a huge modern city with paved streets and gray buildings, surrounded by a beautiful desert. Each child's mental picture is profoundly different, yet both sit well within the possibilities of this description.

Similarly, scripts and other actions in MOOSE Crossing are described exclusively in words. Players regularly see messages like these as they interact with other players, their creatures, and their scripts:

```
You pet Rover.
Rover wags its tail and licks your hand.

You climb out of the fireplace and into Randy's room...

Hooloo waddles over to Sam hoping to get something to eat.

A glint of metal catches your eye.  Poking through the
velvety grass is the corner of a large machine.  You dig it
up and polish it and standing before you is a glistening
time machine.

It is late-afternoon rainy season. There is a small break
in the rain right now...so hurry about while the weather
lasts...
```

Much like reading a good novel, users create rich mental images of the things, places, and people they meet in the virtual world by reading these kinds of text descriptions.

On the other hand, a graphical world presents a specific image for the backdrop of the world, and explicit drawings for objects and characters in the world. There is no need for the kind of imaginative mental images that run rampant in MOOSE Crossing—the graphical nature of the environment lays it all out. This limitation of graphics can also be seen as a big advantage. For some users, having to read so much text limits their understanding of what the virtual world really is, and they frequently overlook key features as they skim through room descriptions and other messages. It's much easier to figure out what's going on in a graphical world—it's all there in front of you, all the time.

One major reason for this is that graphics support multi-modal kinds of expression. Unlike text-based worlds where all information is provided via a single channel, a text stream, graphical worlds make use of many different modalities of expression, including visual cues, sounds, and spatially separate output areas. Users can interpret the location, intention, emotion, action, and other features of a character just by looking at the character, noting which way it is facing, whether it is smiling or grimacing, and what it is currently doing. As characters talk, their words hang in speech bubbles above their heads, clearly indicating who is talking.

In text-based worlds, an object's location within a room is generally hazy; usually there's no information about who is standing next to whom, whether your character is leaning on a door or sitting on a chair, nor is that kind of information generally important. Conversely, in a graphical world, there is much more emphasis on location and action. A character's location on the screen and its visual appearance are fundamental ways for users to express themselves. Animations are eye-catching ways to display a character's actions, emotions, and intentions. At Pet Park, kids move their characters around so that they stand on planets, or are climbing up a set of stairs. A kid might scribble her character around the screen to get someone's attention, or to relieve frustration. Kids have their characters laugh when someone says something funny, or cry with mock-grief when erasing an old creation to make room for a new one.

Graphics make it easier for lots of objects to be acting and interacting with one another in a way that users can understand what's going on. For example, consider this hypothetical scene in a Pet Park room. A dog is chasing a bird around a room in Pet Park, and at the same time, the bird is gathering food for her chicks, which are rustling in a nest high in a tree. Meanwhile, two players are talking in a corner of the room, showing off their new scripts to one another, and pointing out the bird's script as it flies by. In a graphical environment, it's easy to comprehend this kind of situation at a glance since each graphic element is separated visually. Furthermore, it's easy to filter out parts of the action to focus on others. A user may ignore the birds to pay more attention to her conversation with a friend, or focus on testing a new script for her dog, ignoring the moving scenery entirely.

However, this much "motion" in a text-based world would be confusing at best, and would probably be incomprehensible to any but the most experienced MUDder. Text messages would get interleaved with one another, making it hard to follow the train of a particular script. The sheer volume of messages generated by the actors in the scene would obscure the flow of the conversation. The single text stream makes it harder to filter out the parts of the action the user doesn't care about, while still being able to pay attention to the rest.

3.2 Accessibility

For early readers, a text-based world like MOOSE Crossing is hard to comprehend. Words fly by a mile a minute, and it's tough to know which parts are important to read, and which aren't. Early readers labor through every word, but seldom understand what they've read at the end. As the room description scrolls off the page, they get confused about where they are, yet have trouble remembering the key commands like "look" and "say" that would help them reorient themselves. Programming in a text-based world requires a lot of precise spelling and attention to detail that is difficult for kids who are still learning how to read and write.

Graphical worlds avoid a lot of these problems. First, there is very little text, if any, that kids need to read. They can get around by using buttons, menus, and other graphical interface widgets, which also serve as a constant, visual reminder of the available commands. Second, the room display doesn't scroll away like its text description would in a text-based world. It's always obvious where your character is, what room you're in, and who else is nearby, since that information is always displayed in the room view. Third, talking with other users can be achieved with a microphone and a set of speakers, and doesn't necessarily require typing a message with the attendant spelling and sentence-construction difficulties.

Furthermore, worlds with graphical programming environments can help lower the age threshold for making new behaviors for one's creations. Users can program their creatures by stringing together a few commands, represented by graphical building blocks with iconic labels. A graphical programming language makes it easier for kids to write programs without having to worry about getting the spelling and syntax right. However, graphical programming can be more limited due to the difficulties in expressing more complex programming structures in a graphical display. Because of this, graphical programming languages are best used as stepping stones on the way to introducing more expressive textual languages.

3.3 Expressiveness

The richness of expression possible in MOOSE Crossing is due, in part, to the nature of the text-based world. Through writing descriptions and programming scripts, kids can build whatever they can describe in words. Creatures, objects, rooms and their scripts can do and say just about anything, all described with a few well-articulated sentences. Kids perfect their descriptions and scripts by editing their words to produce high-quality writing and imaginative imagery. Anyone who has a reasonable command of the English language and a good imagination can build with a tremendous amount of expressive power in a text-based world like MOOSE Crossing.

In a graphical world, however, creation comes in many forms, each with varying levels of expressive power. The most obvious, and most expressive, way for kids to build in a graphical world is by drawing new creatures and other objects. However, this method of creation is also the hardest. It's easy to draw a scribble, perhaps even easier than writing a sentence. Unfortunately, it is somewhat more difficult for a budding artist to produce a high-quality image with a comparable level of detail as the descriptions kids wrote in MOOSE Crossing. The mechanics of drawing on a computer are hard and time-consuming, even for adults. Going the next step to draw several frames of animation for a new creation takes even more time and skill, and requires the support of good graphic design tools. The attention to detail and careful staging that was possible with text descriptions can be limited in a graphical world by users' drawing talent and patience as much as by their ideas.

However, there are other ways of allowing kids to create things in graphical worlds beyond drawing. One alternative is for kids to give textual descriptions to their objects, writing descriptions for their creations similar to the descriptions that kids wrote in MOOSE Crossing. Of course there are limits to this kind of expression—descriptions must still correspond in some way to their associated images, or else they won't make sense. Another richer alternative is for kids to make use of the pre-drawn objects in the world, but provide tools for kids to create new behaviors for these objects through writing programs. Kids can personalize their creations by teaching them a unique set of behaviors, using scripts to combine the behaviors that come with the pre-built objects. Other object creation models where kids can create their own characters by connecting mix-and-match body parts together are also promising ways for kids to create without necessarily having to draw.

In graphical worlds, it's possible to let kids create at a variety of levels, each providing different degrees of expression. At the simplest level, kids can create by copying the graphics of existing objects, giving them new names and text descriptions. Kids can go further to define new behaviors for their creations by writing scripts. And, for kids who have the talent and patience, they can create their own graphics and animations from scratch, lending the most expressive power.

For the reasons discussed above, kids are likely to use pre-drawn objects in their creations in graphical worlds. Hence, these worlds will tend to be populated with many similar-looking objects, limited by the extensiveness of the library of pre-built objects.

This leads to an important design consideration for graphical worlds. The environment needs to help foster personal attachments between kids and their creations, since a child's creations will be less distinguishable graphically from others' creations. Encouraging an online culture in the community that values creative scripts is one way of approaching this. Another related consideration is that kids who opt to draw their own creations might spend so much time perfecting their drawings that they miss out on the programming aspects of the world. While both the drawing and programming activities have merit, the designer of a graphical world may choose to highlight one or the other based on the educational goals for the project.

3.4 Programming Paradigms

Not surprisingly, the medium changes the kinds of programs that kids want to make and the types of commands that are used most frequently. In MOOSE Crossing, most kids' programs make heavy use of commands that printed messages to the screen:

```
tell player "Here is the magic number: " + my magic

say "Welcome to the Toy Shoppe!  Type 'buy toy' to buy a
toy!"

announce_all_but context context's name + " scratches Rover
behind the ears."

emote "laughs and jumps up and down with glee."
```

In comparison, most kids' programs in Pet Park consist of calls to building block animations, and make heavy use of the 'repeat,' 'wait,' and 'say' commands:

```
spinHead

repeat 10 [laugh]

say "I am the first chicken in space."

wait 2
```

Programs in text-based worlds are centered around printing sets of words in various ways, and to various people. In comparison, programs in graphical worlds move avatars around the screen and animate them to do behaviors. Scripts in graphical worlds do use text to print messages, but generally only as a supplementary output mechanism along with movement and animation.

Because repetitive actions are interesting to watch and don't interfere with other activities, programs in graphical worlds are more likely to describe a continual movement, such as a ball that bounces off of the walls of a room repeatedly until someone catches it. In text-based worlds, this kind of continual action is discouraged because it makes normal conversation in a room confusing due to frequent and incessant messages. Graphical worlds are also more likely to see several objects doing their behaviors in parallel, for similar reasons.

3.5 Learning Potentials

In text-based, constructionist worlds like MOOSE Crossing, a major part of the learning experience includes practice in reading and creative writing, as well as computer programming. As kids build creatures, rooms, and other things, they give them rich, textual descriptions, and teach them behaviors by writing computer programs, or scripts, for their creations. In the process of building, kids learn powerful programming concepts that are rarely taught in schools before the high school or college level.

Graphical worlds can provide similar learning opportunities in computer programming as text-based worlds. However, graphical worlds naturally de-emphasize the reading and writing that is so central in text-based worlds. Instead, graphical environments compensate by bringing in a whole new discipline: graphic design. Kids designing in graphical worlds learn how to draw creatures and how to animate them. They learn by example, at first using the sample creatures, then later making small graphic changes to the samples to give them new behaviors, and eventually drawing their own creatures from scratch.

Graphical worlds can also help present certain concepts in an intuitive way. Graphical programming languages, although sometimes limited in expressive ability, help concretize and make abstract programming concepts more understandable, especially to novice users. Much of the added learning potential in a graphical world comes from being able to engage younger kids in constructing the virtual world, scaffolding their learning as they begin to write programs first in a graphical language, and later in a textual one.

3.6 Practical Matters

On a more practical note, graphical worlds need a lot more computational power, and frequently need a faster network connection, than text-based worlds. Most text-based worlds are classic single-server systems, where all client traffic is routed through a single server over the Internet. Clients for text-based worlds range from simple telnet connections to relatively small, simple client software packages that implement a windowing user interface. At either extreme, today's lower-end computers, with standard modem-quality network connections can run these clients comfortably.

In comparison, the client software for graphical worlds needs to not only coordinate and synchronize information with other clients, but it also needs to display graphics and changing animations. A graphical image contains orders of magnitude more data than a paragraph of text; hence, graphics require both faster processors and faster network connections to manage. Furthermore, because of the extra computational load brought on by the graphical medium, single server system architectures no longer work for graphical worlds. Distributed system architectures are used to shed load from the servers, placing an even greater computational load on individual client computers. Some graphical systems are designed to use multiple servers; others experiment with entirely serverless systems. No matter the architecture, graphical world clients need access to much more computational power than textual world clients, and depending on the architecture, faster connections to the Internet.

Although some home and school users do have access to higher end computers capable of running the software for graphical worlds, most home and school computers do not. This leads to a natural stratification in user backgrounds for graphical worlds. For the moment, graphical worlds are populated with users who can afford the latest and greatest computers—typically those from more prosperous families, living in richer school districts and communities. Although this will change over time as computers inevitably continue to get faster and cheaper, graphical worlds will also continue to expand in scope to use that added computational power.

4 Criteria for Designing a Constructionist, Graphical Online World for Kids

Over the course of this project, a few larger ideas have emerged about how to go about designing constructionist, graphical online worlds for kids and other novice users. This section presents a set of criteria for future designs, based on my personal experience with the Pet Park prototype and the MOOSE Crossing project, and informed by an understanding of the main affordances of graphical worlds as discussed in Chapter 3. Not surprisingly, many of these criteria could extend to other kinds of online worlds as well.

There is a big picture behind these criteria. Simply put, it is a balance between designing the building environment and supporting an online community. A carefully designed building environment without a strong community will miss many of the benefits of online community support, including access to role models, an audience of peers, added motivation by seeing others' projects, and a network of others who can help with problems. What good is a multi-user environment if the inhabitants don't make use of each other's talents and resources? On the other hand, a strong community within a poorly designed building environment doesn't work either. Online communities of kids help each other over stumbling blocks in their projects, but if the stumbling blocks are too large or too numerous, only a minority of users in the community will ever figure out how to get past them. It's unreasonable to expect the community to compensate for a poorly designed, difficult-to-use building environment. Good environment design and good support for community development must go hand in hand.

4.1 Provide Multiple Levels of Access, and Bridges between Levels

A strong undercurrent of projects like MOOSE Crossing and Pet Park is the idea of designing technologies that are accessible, yet have a high ceiling. Kids can get started easily, but the possibilities are open-ended enough that they don't feel boxed in, and have lots of room to grow their skills. One design philosophy that addresses this goal is to have a learning curve that starts close to ground zero so that it is accessible, and a learning curve gentle enough in slope that kids can progress without running into major hurdles. This notion of a learning curve suggests that learning is a smooth process, each step along the curve leading imperceptibly to the next. Yet learning is seldom so smooth—there are inevitably bumps and hurdles to overcome along the way.

Another way of looking at the learning curve metaphor is as a set of levels of access, and bridges that help kids move between those levels. Each level provides a learning experience along some dimension, and is compelling and interesting for the user. Levels are a way of managing the tradeoffs between making an environment accessible, yet providing a high ceiling. Earlier levels need to be accessible to the most novice users, yet can afford to be less expressive along some dimensions. Later levels can require users to be a bit more sophisticated in order to use the more expressive tools they desire. This isn't a problem, however—by the time users get to the higher levels, they have developed the skills they need to navigate this more complex space. Levels help scaffold the user's learning experience by broadening the tools available to them as they move to the next level. By no means are kids done with the environment when they reach the highest level—rather, that is the point at which they finally have full command of all of the expressive tools available and can build with a maximum of expressive freedom.

Of course users won't necessarily progress from one level to the next as neatly as the designer might like, absorbing everything on one level before proceeding to the next. They'll jump around somewhat, learning various tidbits from friends they meet online, incorporating them into their projects as they go along. Overall, however, their progress and understanding will follow a general trajectory from the simplest concepts and projects to more complex ones, from the first level to the highest. Users won't know what level they're working on, nor is that information important. Rather, the idea of levels is more of an aid to the designer, a check to make sure that each level is interesting and meaningful on its own, and that bridges connect levels in a logical order, encouraging kids to move up the learning curve. A design that makes use of this analysis is less likely to overlook big hurdles in the learning curve that will halt kids' progress.

The first level should be optimized for the youngest, most novice user. Activities targeted at this level need to have the most streamlined user interface and the most obvious online help. If the youngest target user is a pre-reader, this first level should clearly not require any reading or spelling. Paying close attention to this first level is crucial for two reasons. First, it is important to capture new users' interest right away. An early stumbling block can turn new users away before they even get started, especially if no human kids happen to be around to help. Second, the kinds of activities and functions targeted at this level are likely to form the basis of functions for later levels, so it's important to make them intuitive and easy to use for everyone's benefit.

Subsequent levels should build on each other in a logical progression, always making sure that there is a clear path from one level to the next. Later levels may introduce text as a supplementary medium, allowing text descriptions for characters or providing context-sensitive help pointers that help kids discover new functions as they go along. Limited screen real estate and menu space will inevitably dictate that some commands used in the more advanced levels be more hidden from sight than other commands used in the first few levels. Again, this is acceptable because users who are working at the higher levels will have gained enough familiarity with the system to manage the more complex user interface.

4.1.1 Designing by Levels: An Example

This subsection will describe in some detail the levels for a world like Pet Park, to help make the levels design concept more concrete. Certainly there are many ways of delimiting the levels of a design—this section presents just one such arrangement. The levels in this design are not of equal size and scope. Rather, they are conceptual groupings. Most users will probably fly through the earlier levels, but spend considerable amounts of time at each later level. Younger users may take longer on lower levels if they are not ready to move on, perhaps because they haven't yet learned to read and write. This design is intended to show how kids can grow along with an environment like Pet Park, progressing to the higher levels only when they've both exhausted the previous levels, and are developmentally ready for the extra challenges. This design also puts a high emphasis on kids being able to create without necessarily needing to draw the graphics for their objects.

The first level of a world like Pet Park includes the most basic mechanics of talking with other characters, moving a character within a room, and moving between rooms. At this level, kids can explore the virtual world and talk to other characters. The interface provides graphical and audio cues when new users first log on, showing them how to talk to other characters and move around. Functions at this level need to be accessed with point-and-click style commands, such as dragging with the mouse to move a character around the screen, double-clicking on an exit in order to get to a new room, or holding down a microphone button to speak a message into the room.

The next level includes discovery and customization of one's own character, including running your character's pre-built behaviors, and defining new behaviors by combining the pre-built ones with a simple programming language. This level also includes making new things by making children of existing objects in the world, defining new behaviors, making new rooms, and arranging objects in rooms by moving, dropping, and taking them.

A simple, graphical interface to the programming language is crucial at this level. Early readers and other novice users will benefit greatly from being able to get their feet wet by writing simple programs in an environment where spelling and syntax doesn't count. The graphical interface doesn't need to include every available command in the language; rather, it should highlight a core set of the commands that kids would want to use in their scripts. This core need not go beyond the character's building block behaviors, a handful of sound effects, a repeat statement, and a say command. On one side of the interface window is a workspace where your program resides. On the other side is a menu of building blocks that you can drag over to use in your program. Puzzle-piece connections between command blocks make it obvious which blocks can connect to each other, and in what way.[15]

[15] For more about the graphical programming interface being developed in tandem with Pet Park, see (Cheng 1997).

Another important feature of this level is naming new objects, and some method of identifying them uniquely. The original Pet Park prototype made use of a global namespace where names had to be unique, with no spaces allowed in object names. Many kids got frustrated that others had already picked all the obvious names like "door," "couch," "chair," and "dog," yet they couldn't use names like "big brown dog" or "yellow chair" because of the restriction on spaces. After a few rounds of error messages, the kids began to pick nonsensical names for their objects like "cc," "g," and "exit345." The environment needs to allow kids to put spaces in objects' names to help alleviate this. Furthermore, there's a better way to guarantee name uniqueness without cutting down the namespace so severely. First, guarantee uniqueness of kids' usernames, and then have the formal name of every object include the name of its owner, as in "Taffy's couch" or "couch (by Taffy)." That way, each kid essentially has a separate namespace, yet names still remain unique on a global scale.

The third level introduces the textual programming language. This is the first big hurdle in the kids' progress so far. Luckily, there is a nice bridge that both encourages kids to discover this level, and helps them make sense of the new language they are presented with. A button on the programming interface window will switch the current program back and forth between "graphics mode" and "text mode." At some point while making a program, kids will press the button, either by mistake or just to "see what it does." Switching to the text mode will translate the current graphical program into a corresponding textual one. Kids can switch back and forth, seeing how each element is translated, discovering that the two representations for their program are equivalent. In text mode, the menu of building blocks in the program workspace is now much more extensive, including commands from the entire programming language. Kids can discover the new commands by exploring the expanded building blocks menu, and can get help on the new commands at the touch of a button. Commands entered in text mode that are not directly supported by the graphical interface can be translated into the graphics mode. They'll appear as basic block shapes with the command inscribed, but won't be editable until translated back into text mode.

The last level allows kids to modify the graphics of their objects with a graphics editor to edit individual frames of their objects' animations, stored as properties on that object. Up until now, kids have been using objects' pre-existing building blocks in their creatures' behaviors. At this level, they can create new behaviors for their objects by modifying graphics on the existing building blocks, and even create brand new objects with brand new graphics. This level introduces the tools that let kids modify existing animation frames, and create new drawings from scratch.

4.2 Start with Good Graphics, and Provide Good Graphic Tools

It seems obvious that a graphical environment needs to have good graphics in order to be compelling for its users. But what differentiates good graphics from bad ones? Of course they need to look nice and be appealing to a wide age range, but there is more to it than just looks. If we expect kids to add their own drawings to the virtual world, we need to make it possible for kids' drawings to be able to mimic the theme and style of the graphics already in the virtual world. Otherwise, there is a strong incentive for kids not to bother drawing their own graphics—their work would never compare to the quality of the existing drawings, and would always look funny next to the professional ones.

There are two, highly interrelated ways to address this concern. One is to pick a graphic style and sample graphics that are easy to copy and modify. For example, characters with simple shapes and solid colors are a lot easier for novice artists to work with than ones with fancy 3-dimensional shading. However, it would be a mistake to have the quality of the professionally drawn graphics toned down to meet the level of kids' drawing ability. Rather, the sample graphics should provide a high-quality example for kids to use as models as they do their own drawings, yet still be achievable with the available tools and ample servings of practice and patience.

The other solution calls for a set of special-purpose graphics tools that enable and empower kids to draw in the style of the existing art. The graphic tools need to support both drawing new creations and animating them. A special-purpose tool geared towards the chosen graphic style would help kids make creations that blend in with the existing world. For instance, having the tool automatically draw black outlines at color boundaries helps maintain a cartoon character style. Making it easy to make a change to all of the frames of an animation concurrently is another crucial feature. Even very simple modifications, such as coloring a chair purple instead of yellow, or putting a spot on a dog's belly, would be tremendously time-consuming if the change had to be made by hand to each frame of that character's animations. Clearly an environment needs both a good graphic style, and good tools that help kids create in that style.

There are other ways of making it easier for kids to make their own drawings as well. "Divide and conquer" is a popular ideology in many intellectual disciplines, and can be applied here as well. Instead of having the world composed of complete animations, each character or other creation in the world could be composed of parts. For instance, a dog might be composed of a head, a body, a tail, and feet, each individual images that are glued together to form a whole character. Each part is drawn and animated separately, and parts can be mixed and matched with other parts to create a large variety of creatures from a relatively small library of parts. Because they are smaller and less intimidating, it's easier for budding artists to draw and animate one part at a time, and then connect them together in the end to form the whole.

4.3 Develop for a Learning Community

A constructionist, building environment benefits greatly from an online community that takes pride in helping others, eagerly shares their creations, and supports new members. Users of an online, building world form a community of people who are both learners and teachers, people who are interested in making new things that push the limits of what they've done before, but still take the time to share their knowledge and help others. This all sounds great on paper, but how does a designer encourage this kind of community to develop and flourish?

Many researchers consider this an open question—no one quite knows how communities develop the way they do, what influences their growth, or how they're different from more conventional, geography-based communities. However, there are ways to stack the deck, so to speak, to help encourage and direct the growth of a community so that it will enrich the entire learning environment. The designer never has total control over the community—the best she can do is provide tools that help the community grow in positive directions, and provide leaders early on during the community's formation to help influence its growth.

Hence, when designing an online environment, it's important to keep the community in mind. Every aspect of the design should be evaluated to see what kind of an effect it might have on the developing community. Issues of privacy, anonymity, and users' ability to restrict access to their own objects all affect the growth and direction of a community.

At the very least, kids need tools to help find each other and be able to converse online. The user interface needs to make it easy for kids to talk with each other, streamlining the commonly used communication commands. Functions that help users find each other are crucial as well. As the virtual world grows to include tens and hundreds of rooms, it's easy for many users to be working online concurrently, yet never run into each other. A potential solution borrowed from text-based worlds is the "who" command, which lists the players currently online and their locations in the virtual world. Supporting tools that allow you to join a particular player on the who list are important as well. Additional tools that help encourage community growth, such as in-world email and mailing lists also help community members forge relationships with each other and become more active participants in the community.

For graphical, constructionist worlds like Pet Park, we'd particularly like to encourage a learning community that supports each others' learning experiences. Hence, it's important to make decisions regarding privacy, locking rooms, and user anonymity in the interest of developing a trusting, learning community. The environment should be open to promote sharing ideas, yet not take away kids' control over their creations. The entire virtual world should act as a library of examples. Kids need to be able to inspect and learn from every object that they find online. Locked doors inhibit this kind of exploration and sharing. Anyone should be able to view an object's scripts and properties, however, the owner needs to retain control over whether others are allowed to use his creation's scripts, or whether others may move that object around.

In graphical worlds like Pet Park, public and private spaces are community-building tools. To form strong virtual communities, members need to feel like they belong. Encouraging community members to make their own rooms—a private space where they can keep their treasures—helps people feel like inhabitants of the virtual world, not just visitors. On the other hand, public spaces are just as important. Having new places to explore, neutral places to meet other members, and places to hold gatherings helps build community by bringing people together.

5 Future Directions

Experimenting with Pet Park has indicated that graphical virtual worlds can indeed be viable building spaces for novice users, and that they hold promise as powerful learning tools. Graphical worlds like Pet Park can be compelling and interesting places in which users play, work, and build. My experience working with both Pet Park and MOOSE Crossing has helped me develop a deeper understanding both of the textual and graphical media, and of their relative affordances and constraints. Through this study some basic design criteria for graphical virtual worlds have emerged, paving the way for the next generation of constructionist, graphical virtual worlds.

However, like any good research, this project has raised many more questions than it has answered. What programming language should be used for these new graphical virtual worlds? How can we better take advantage of the graphical medium to elucidate programming concepts and make the programming language interface more intuitive? Can kids learn as much from graphical worlds as they can from text-based worlds? How does the graphical nature of the world affect the community that develops? How do you design tools to help novice artists draw and animate their original creations? How can sound, video, and other multimedia best be incorporated into graphical worlds? Can we design a system architecture that supports hundreds or thousands of users simultaneously? Can these system architectures scale to support tens of thousands of users, or is there a limit? What about using 3-dimensional graphics instead of 2-d graphics? Can the 3-d graphical medium also be used successfully for multi-user, constructionist learning spaces? The questions go on and on.

Pet Park is a very early experiment in a young, developing medium. There is much yet to learn about graphical, constructionist learning worlds; Pet Park represents only the tip of the iceberg.

Bibliography

Begel, Andrew. "Bongo: A Kids Programming Environment for Creating Video Games on the Web." Master of Engineering Thesis, MIT, 1997.

Begel, Andrew. "Bongo Home." MIT Media Lab, Epistemology and Learning Group, 1997. http://el.www.media.mit.edu/bongo/

Bruckman, Amy. "Cyberspace is Not Disneyland: The Role of the Artist in a Networked World." Commissioned by the Getty Art History Information Program, 1995. http://www.ahip.getty.edu/cyberpub/bruckman.html

Bruckman, Amy. "MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids." Ph.D. Thesis, MIT, 1997.

Cheng, Andrew. "Pet Park Blocks: A Graphical, Programmable Environment for Animating Agents." Master of Engineering Thesis, MIT, 1998.

Dyson, Esther. *Release 2.0: A Design for Living in the Digital Age.* Broadway Books, New York, 1997.

"Palace Support Online Manuals: Iptscrae Overview." The Palace, Inc, 1997. http://www.thepalace.com/support/manuals/iptscrae/ipt-2.html

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas.* BasicBooks, New York, 1980.

Papert, Seymour. "Situating Constructionism." In *Constructionism.* Eds. Idit Harel and Seymour Papert. Norwood, NJ, Ablex Publishing, 1991.

Resnick, Mitchel. *Turtles, Termites, and Traffic Jams.* Cambridge, MA, MIT Press, 1994.

Rheingold, Howard. *The Virtual Community: Homesteading on the Electronic Frontier.* HarperCollins, New York, 1993.

Turkle, Sherry. *Life on the Screen: Identity in the Age of the Internet.* Simon & Schuster, New York, 1995.

Turkle, Sherry. *The Second Self: Computer and the Human Spirit.* Simon & Schuster, New York, 1984.