Playground: An Object Oriented Simulation System with Agent Rules for Children of All Ages

Jay Fenton
Kent Beck
Apple Computer Vivarium Project
292 S. La Cienega Blvd.
Beverly Hills, CA 90211

Abstract

Programming languages for children have been limited by primitive control and data structures, indirect user interfaces, and artificial syntax. Playground is a child-oriented programming language that uses objects to structure data and has a modular control structure, a direct-manipulation user interface, and an English-like syntax. Integrating Playground into the curriculum of a classroom of 9- to 10-year-olds has given us valuable insights from the programs intended users, and confirmed many of our design decisions.

Introduction

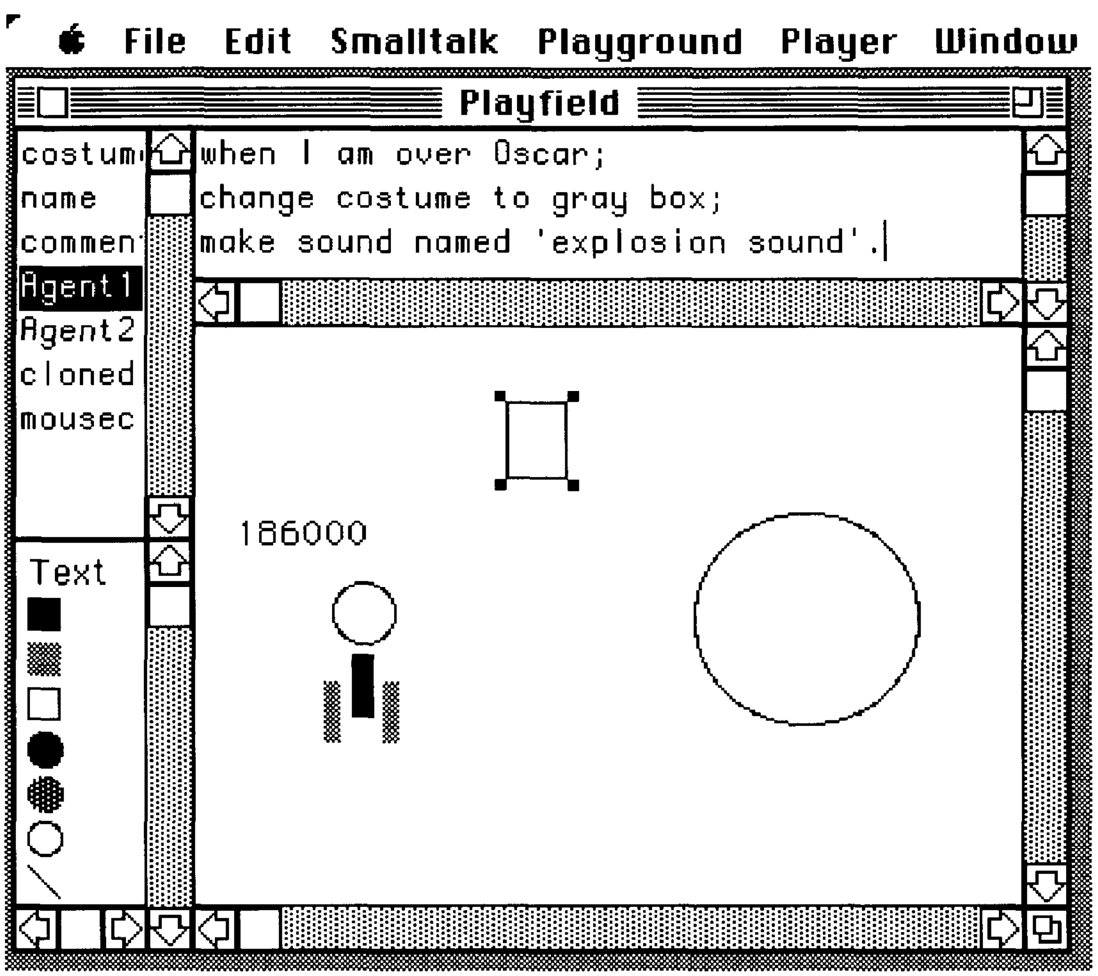
The Apple Computer Vivarium Project was started in 1986 by Ann Marion and Alan Kay and represents a broad research initiative to investigate the phenomena of learning. To provide a living laboratory for study and experimentation, Apple established a relationship with the Open School, a public school in the Los Angeles Unified School District. As part of this research program, we have created the Playground programming system.

Playground is an object oriented programming environment that allows children to construct simulations by endowing graphical objects with laws to obey. Playground is inspired by our intuition that biology provides a good metaphor for understanding complex dynamic systems. Children will write programs by constructing artificial animals and turning them loose in an environment. Each object is a separate creature, with sensors, effectors, and processing elements, that can act of its own accord.

Our exposition begins with a demonstration of Playground as it would be experienced by a new user. The language is then presented in terms of examples. Next, we consider the influences that led us to adopt agent rules as our unit of computation. The following section deals with implementation details: how we make agent rules work and how they make animation and communication easy. We then recount our experiences teaching this language to children and conclude with our ideas for future directions.

Overview

The basic elements of the Playground environment are illustrated by the program's screen display as shown in figure 1.



```
when I am over Oscar;
change costume to gray box;
make sound named 'explosion sound'.
```

Figure 1: Playfield Overview

When first started, Playground behaves like an object oriented drawing program, permitting the user to construct pictures using geometric primitives. Circles, squares, bitmaps, text, and composite objects can be placed anywhere on the screen. Any object or collection of objects can be selected and manipulated through menus. Objects can be opened up and their constituents browsed.

In Playground, objects occupy a planar surface called the *Playfield.* This Playfield can be viewed as a world inhabited by organisms. Each organism in the environment is a *Playground Object.* The Playfield mediates the interactions between the objects within it. Any object can be opened up and investigated, becoming itself a Playfield with constituents. To introduce an object, select a prototype from the gallery of predefined objects. Then click on the Playfield to introduce a copy. This object can then be selected, moved, or resized. When an object is selected, the editing area above the Playfield becomes active, permitting the user to edit the agent rules.

Agent rules describe cause-and-effect relationships that apply to the simulation. When an appropriate set of circumstances comes up, the agent rule is triggered and the designated sequence of operations is followed. Agent rules run in parallel.

As the rules execute, they can move an object or change its appearance. The animation code reacts to these changes by repainting the screen as needed to achieve a real time presentation.

Agent rules are entered using the *caption pane,* which is displayed above the Playfield. The appropriate agent name is selected in the *agent name list pane* on the left. The caption pane applies to whatever object is selected on the Playfield.

In figure 1, the user has selected Agent1 for the square object. This is indicated by the system highlighting the agent name in the pane on the left, and drawing small rectangles called "birdies" at the corners of the selected object. By moving the birdies the user can resize or reshape the object.

Each organism controls how it is presented to the outside world. This is done by donning a *costume*, a generic shape such as a circle or square, or a bitmap graphic. A costume defines both the physical appearance of an object and how it interacts with the user. For example, graphical objects can be resized, while text objects can have their font changed. An object can also move, change size or color, font, etc. under control of agent rules.

An object can sense the presence of other objects on the Playfield in various ways. Playground provides functions that return sets of objects that are nearby, are of a certain type, that overlap, and so on.

All Playground objects may avail themselves of a background of predefined behaviors which implement a useful naive physics [Gardin89] of location and motion over time. Each object has a heading and velocity which control motion across the Playfield according to the rules of turtle geometry as defined in the Logo programming language [Papert80].

Language

In our experience with children, we have found that a surface syntax that closely resembles that of a natural language makes teaching a programming language easier.

Grammar

Playground is defined by a phrasal grammar that uses a syntax closely resembling that of a natural language.

Each Playground clause corresponds to a "message send" in a conventional object oriented language. The user program is parsed according to these phrasal grammar rules, which then generate Smalltalk 80 code for compilation. References to undeclared names are allowed, and are resolved at run time using a dynamic binding function.

Here are some sample Playground sentences:

```
Change costume to black box.
Move arrow to 30 @ 50;Make sound 'loud growl'.
```

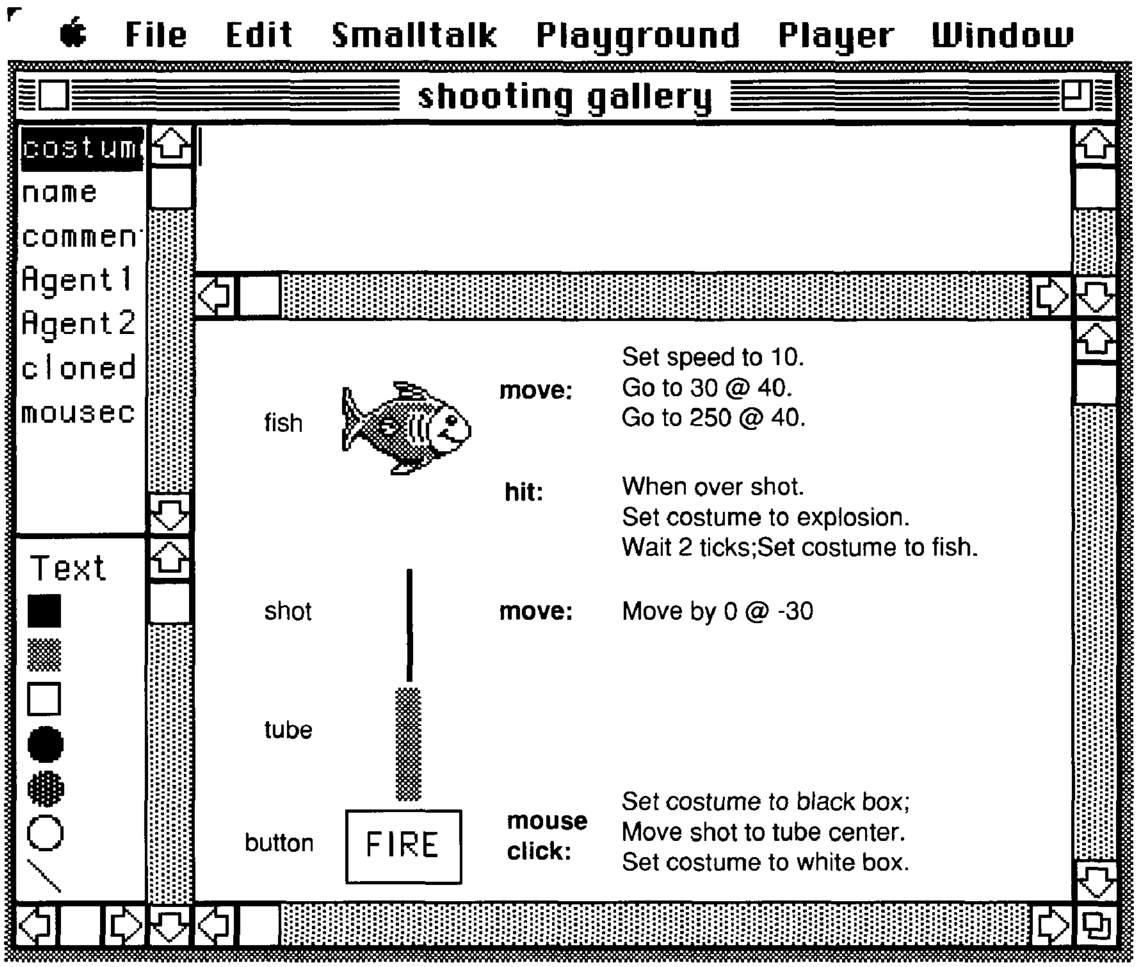
In the first example sentence the current object, or *self*, is commanded to change its costume into a black box. In the second sentence, the object named "arrow" is commanded to move to the given coordinate, and then the 'loud growl' sound is triggered.

The semicolon serves as the non-yielding statement separator. The period designates a yield point. During each simulation cycle, every active agent runs up to the next period. Thus process multiplexing occurs at predictable places.

Examples

We have implemented several diverse models in Playground to test its range. Figure 2 illustrates a simple "shooting gallery" video game.

Note that if an agent rule does not specify a condition, it runs continuously.



fish — **move:**
Set speed to 10.
Go to 30 @ 40.
Go to 250 @ 40.

hit:
When over shot.
Set costume to explosion.
Wait 2 ticks;Set costume to fish.

shot — **move:**
Move by 0 @ -30

button — **mouse click:**
Set costume to black box;
Move shot to tube center.
Set costume to white box.

Figure 2: Shooting Gallery Example

Figure 3 shows a predator/prey simulation, with a fixed number of predators.

Figure 4 illustrates a conventional music notation editor with real-time playback. In this example, musical note symbols placed on the staff are triggered by the playback head object in sequence, playing a melody in real time. The sequence can be edited while it is playing.

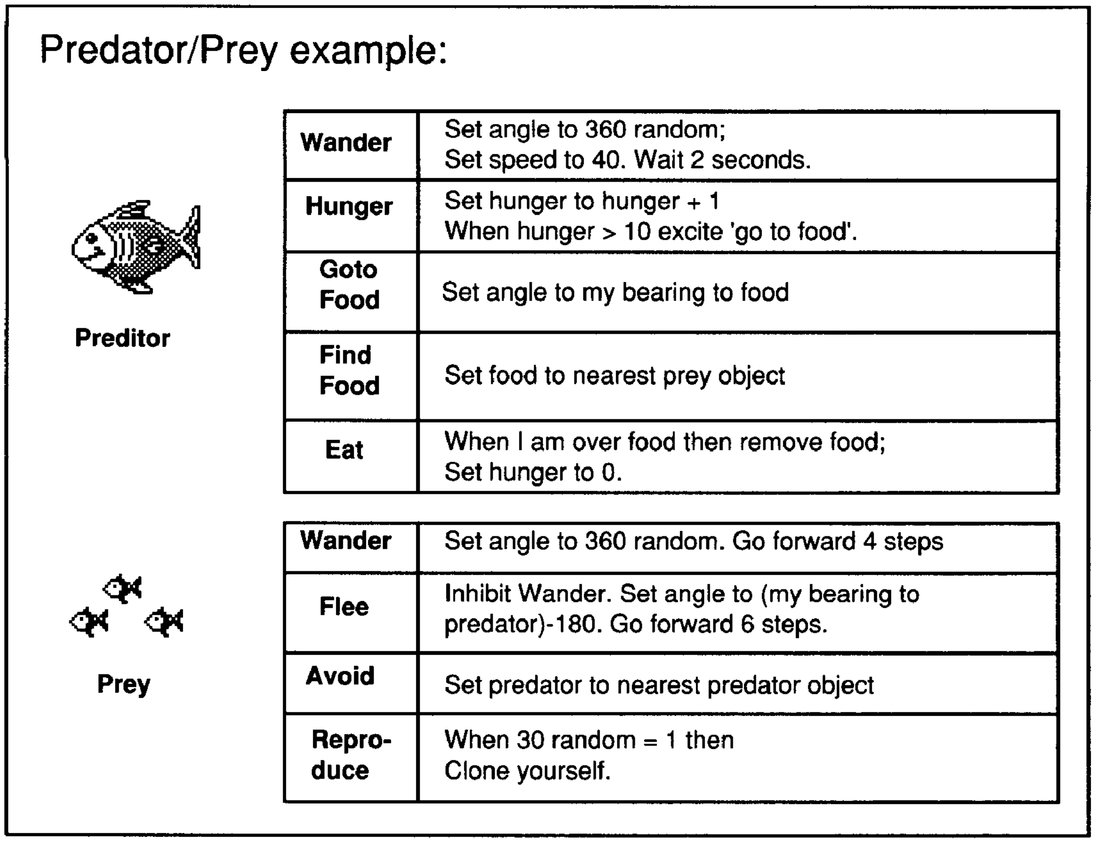
While the previous examples give some idea of the expressive power of the Playground system, to delve deeper we must consider the background influences that led us to this formulation.

Influences

We have studied many fields in searching for the ideas in Playground. A review of these sources of inspiration will help explain the decisions presented in the rest of this paper.

Animal Behavior Models

Since Playground uses the metaphor of biology, it is instructive to study some of the theories of animal behavior that the field of biology offers. Biology is, of course, a vast and fascinating field, riddled with controversy. For a general introduction, see [Grier1984]. We take particular interest in the cognitive mechanisms humans have applied in analyzing animal behavior. This means even ideas that are wrong are interesting, if they give insight into how humans grapple with understanding behavior.

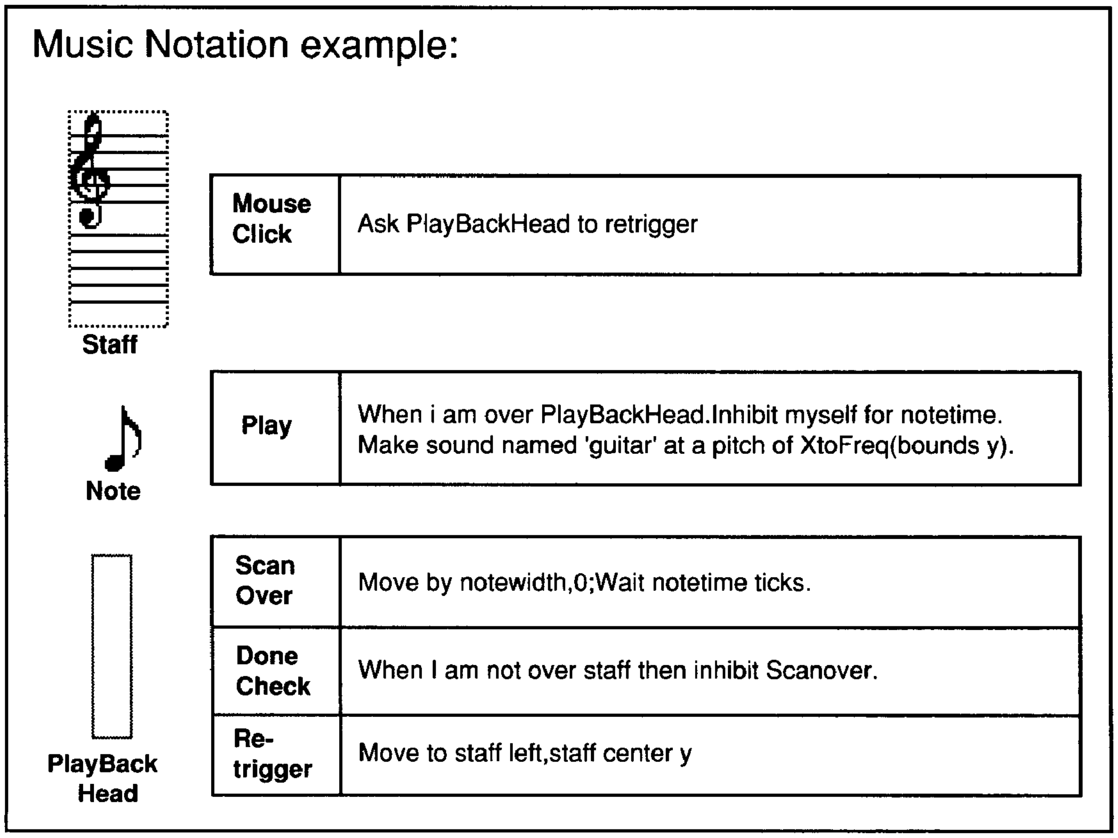


Predator/Prey example:

Preditor	Wander	Set angle to 360 random; Set speed to 40. Wait 2 seconds.
	Hunger	Set hunger to hunger + 1 When hunger > 10 excite 'go to food'.
	Goto Food	Set angle to my bearing to food
	Find Food	Set food to nearest prey object
	Eat	When I am over food then remove food; Set hunger to 0.

Prey	Wander	Set angle to 360 random. Go forward 4 steps
	Flee	Inhibit Wander. Set angle to (my bearing to predator)-180. Go forward 6 steps.
	Avoid	Set predator to nearest predator object
	Repro-duce	When 30 random = 1 then Clone yourself.

Figure 3: Predator/Prey



Music Notation example:

Staff	Mouse Click	Ask PlayBackHead to retrigger
Note	Play	When i am over PlayBackHead.Inhibit myself for notetime. Make sound named 'guitar' at a pitch of XtoFreq(bounds y).
PlayBack Head	Scan Over	Move by notewidth,0;Wait notetime ticks.
	Done Check	When I am not over staff then inhibit Scanover.
	Re-trigger	Move to staff left,staff center y

Figure 4: Music Sequencer

Classical ethology is concerned with observing and describing the behavior of animals. In the classic theory, the organism detects *sign stimuli*, a particular configuration of sensory input, occurring in the environment. These stimuli then influence the organism's innate releasing mechanisms, or *drive centers*. Each drive center has an energy level. These drive centers are organized in a hierarchy, so that drive conflicts can be resolved. The path through the hierarchical tree is determined by the drive centers with the highest energy level.

When a drive center is stimulated and permitted control of the organism; a fixed action pattern is undertaken that "releases" the drive energy. There is a close correspondence between the classical ethological theory and the rule-based expert system approach to artificial intelligence.

$$\begin{tikzcd}\text{Sign Stimulus} \arrow[r] & \text{Innate Releasing Mechanism} \arrow[r] & \text{Fixed Action Pattern}\end{tikzcd}$$

Figure 5: Classical Theory
taken from: [Grier84]

For example, in the three-spined stickleback fish, if a male detects another fish with a red belly assuming the vertical threat posture, the fighting drive center is stimulated. If no higher-priority drive, such as hunger, overrides the fighting drive, the fixed-action pattern for fighting is undertaken. [Tinbergen51]

Society of Mind

The massively parallel organization implied by neurobiology has inspired the "Society of Mind" theory [Minsky85]. Each mind consists of a swarm of communicating agents, each running in parallel, attending to aspects of the problem at hand. These agents are organized hierarchically, and are composed of networks of subagents, the primitive components of which are neurons. Specialized agents attend

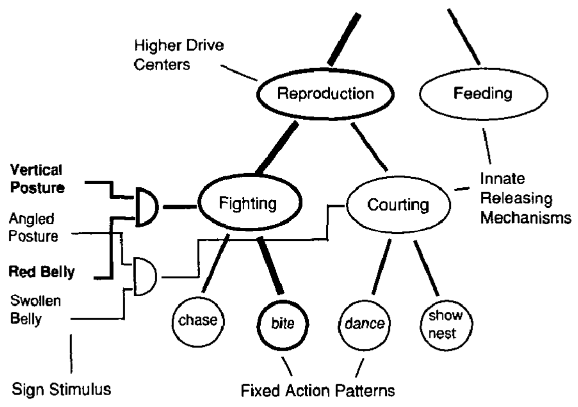


Figure 6: Stickleback Behavior

to memory storage and retrieval, command and control, symbolic processing, and so on.

While the Society of Mind theory has yet to be established by neurobiological evidence, it holds out hope for the unification of symbolic and connectionist theories of Artificial Intelligence. As we propose plausible explanations for how such internal societies must function, we again reveal how the mind attempts to comprehend highly parallel systems.

Agent Rules

Playground seeks to combine ethological and Society of Mind theories by adopting the agent rule [Travers88] as the fundamental unit of computation. An agent rule is an independently executing entity sensitive to particular sign stimuli. This rule can be excited or inhibited by other agents to achieve purposeful control of the organism. When a rule fires, it then triggers a sequence of operations.

Agent rules differ from conventional expert system rules in several ways. Agent rules can be gated by excitation and inhibition mechanisms. They are seen as being strongly situated and are encouraged to use specific references rather than variables. An agent rule may contain a state that persists over time; thus separate instantiations of an agent rule must exist for discrete organisms.

Other Influences
Another influence on Playground's development has been the theory of Idealized Conceptual Models [Fauconnier85] [Lakoff87], or ICMs. In *ICMs*, abstract theories are represented metaphorically using objects and relations grounded in direct experience. We hope, therefore, to be able to model a wide range of phenomena by prefabricating a world with the same properties, operators, and relations as human direct experience, and providing facilities for the construction of metaphors.

There are, of course, a vast array of other influences. The list includes: The philosophy of scientific reasoning and modeling; gaming techniques; ontological engineering; intelligence gathering and analysis; graphical representations of models; audio, film, and video production; air traffic control systems; comic books; magic; and linguistic theory.

Technical Design
Playground has been implemented in C and Smalltalk/V.

Playground objects are universal particles of identity to which any combination of relations or agents can be attached. Instead of having rigid classes, the capabilities of objects are patched together on the fly. This requires a very flexible object system. Every Playground object can contain an arbitrary collection of properties. These can include the source code and compiled code for all locally defined agent rules, the list of active processes for a particular object, animation control tables, and any other attached objects or relations.

Any number of agents can be attached to an object. The code for the agent is either stored locally or reached through other objects that have been designated as examples.

The user can branch from one Playfield to another by programming agent rules. It is also possible to write agent rules that connect one Playfield to another, serving as editors or to characterize functional transformations.
Every Playground object can also be a Playfield if desired. By using appropriate linking instructions in agent rules, Playground can function as a multimedia hypertext system.

The Playfield serves as a bounded universe for causality. Objects on the Playfield affect each other through orderly mechanisms. New forms of causality can be added by creating new types of events and adding agent rules that react to these event messages.

The rules attached to an object can reach across environment boundaries in a limited way. Thus objects can be created which cause general properties to be introduced into the environment they occupy. For example, it is possible to introduce objects into a Playfield which add physical laws that all objects within the Playfield are to obey, in the manner of the Alternate Reality Kit[Smith86].

Serialization, Pushing, and Pulling
A network of parallel agent rules can be simulated sequentially. This is of course required on conventional serial computers, and is useful for parallel systems as well. For example, most fixed action patterns enumerate a sequence of actions that take place over time. While this can be modeled in parallel hardware, it is much more compact and convenient to construct a sequencing automation that triggers each operation in turn. Algorithms can be created to compile parallel dataflow diagrams into sequential instruction streams; see Fabrik [Ingalls88].

There are two fundamental ways to drive a simulation based on agent rules: pushing and pulling. In *pulling*, or event polling, an agent rule tests for conditions explicitly, hoping to detect appropriate configurations in other objects that would enable it to run. In *pushing*, or dependency tracing, when another object generates an event, it notifies the objects that are interested in the change.

Each method has its advantages and disadvantages. Pulling can waste time checking for conditions that have not changed, while pushing requires overhead to track the dependency relationships amongobjects.

Playground supports a mixed mode strategy. Each agent rule can test for the events and situations under which it is to run. When a rule is posted, the condition part is inspected, and—if appropriate—the rule is registered with the system so that it runs only when the appropriate events have taken place. These events can be user interface events, events generated explicitly by other objects, or changes of values for a given object. A program may also trigger an agent explicitly, an operation similar to sending a message in a conventional object oriented language.

Facets

A Playground object is composed of one or more facets. A *facet* is a conventional object with a fixed slot structure that represents a class of properties attached to an identity. The facet idea is an adaptation of the multi-valued relation scheme of KODIAK[Wilensky87]. Some examples of facets used in Playground are: Physical Appearance, Motion, Symbolic Meaning, and Process Header. All identities, including those used by Smalltalk, can have other facets attached to them. Facets help avoid the guilt of having too many NIL instance variables!

All Playground facets share a common superclass of PlayObject. Each facet has a timestamp field, a list of subfacets, and a backpointer to the primary identity this object is a facet for. Each facet kind can add its own fixed and optional slots to this base structure. For example, any object can be given visual appearance, motion, symbolic meaning, etc. when it is convenient to do so, only incurring overhead for such properties when required.

The Playground system comes with a set of predefined facets implementing a wide range of mundane properties of universal utility. These predefined facets can be specialized or overridden as desired. Whenever any value for a slot in a facet is changed, the timestamp field is updated and any other agents that are watching the value of this facet are notified.

Rule Compilation

The Playground language is implemented using a phrasal parser. This design was suggested by the phrasal lexicon theory described by [Becker75].

The language is specified by a large set of shallow production rules which enumerate the phrases the language will accept. There are two kinds of rules: phrasal rules and nonphrasal rules. A *phrasal rule* must contain one or more terminal symbols. A *nonphrasal rule* mentions only abstract grammatical categories.

Playground rules are entered as text strings in the edit pane. Playground converts these textual strings into an array of tokens such as word, number, special character, etc. The terminal symbols in the phrasal rules are matched against the tokens, giving a list of nonterminal symbols to seek..

This list is run down, and successful phrase matches are retained on the parse chart for further analysis. Analysis then proceeds from the top down, seeking a coherent overall structure. The result is a parse tree.
Each phrasal rule contains information on how to convert its meaning into a Smalltalk/V expression. The compiler then transverses the tree, activating the "generate expressions," converting our phrasal syntax into conventional Smalltalk. At appropriate places in the

code, special yield messages are inserted. The result is submitted to the Smalltalk/V compiler,'giving a Smalltalk method which, when executed, generates an instance of a Playground lightweight process.

For example, the Playground agent script:

```
When I am over nest;
Change costume to black box.
Make sound from file 'meow sound'.
```

when compiled against the phrasal grammar:

```
COMMAND ::= when VALUE
        generate: 'self when: &2'.
FUNCTION ::= i am over VALUE
        generate: '(self over: &4)'.
COMMAND ::= change costume to VALUE
        generate: '(self costume: &4)'.
COMMAND ::= make sound from file VALUE
        generate: '(self sound: &5 )'.
NAME ::= black box
        generate: 'BlackBox'.
```

will generate the following sequence of Smalltalk/V code:

```
temporaryMethod
| codeSeq procObject |
codeSeq:= [(procObject semaphore) wait.
        self when: (self over: (self bindingAt: #nest )).
        self costume: BlackBox. Processor yield.
        self sound: 'meow sound'.
        procObject noteDone].
procObject := (PlayAgentProcess new: nil running:
codeSeq).
procObject setAutoRepeat.
^procObject.
```

Whenever a given agent becomes active, this process generator method is executed, which adds a Playground lightweight task to the run list for the host object. These tasks exploit the multitasking capability of Smalltalk/V.

Animation and Communication

Every viewable graphical object has a bounding rectangle, a depth coordinate, and a costume. The bounding rectangle and depth coordinate give the location in the Playfield that the object occupies. The costume points to a costume object, which can be a string, form, primitive shape, or composite. An object can change its costume at will, to animate or to present different editing capabilities to the user. Whenever the costume is changed, the bounds are adjusted to accurately describe the extent of the costume.

One type of useful Playground object is called an *observable collection*. An agent can allocate an observable collection and add other Playground objects to it, and automatically receive notification of any changes occurring to its members. This is how animation and telecommunications are accomplished.

The animation code uses an observable collection to track all objects on the Playfield. When any object changes, it is added to a list of changed objects. During each animation cycle, this list is retrieved and examined. For each object that has changed, the old and new object boundaries are accumulated into a dirty region list of non-overlapping rectangles. For each dirty cluster, the screen is redrawn in back-to-front order in an offscreen buffer, which is then copied onto the user's display screen (see figure 7).

A similar method is used to accomplish telecommunications. Each object to be shared with remote stations is added to an observable collection, which again accumulates a list of those objects that have changed recently. The telecommunications code goes down this list, comparing the present slot values for the object with previous values, broadcasting the changes discovered. The recieving code likewise keeps a list of changes recieved, which it then applies locally.

Both the animation and telecommunications methods require that two copies be kept of each object, one giving the current state, the other representing the previous state.

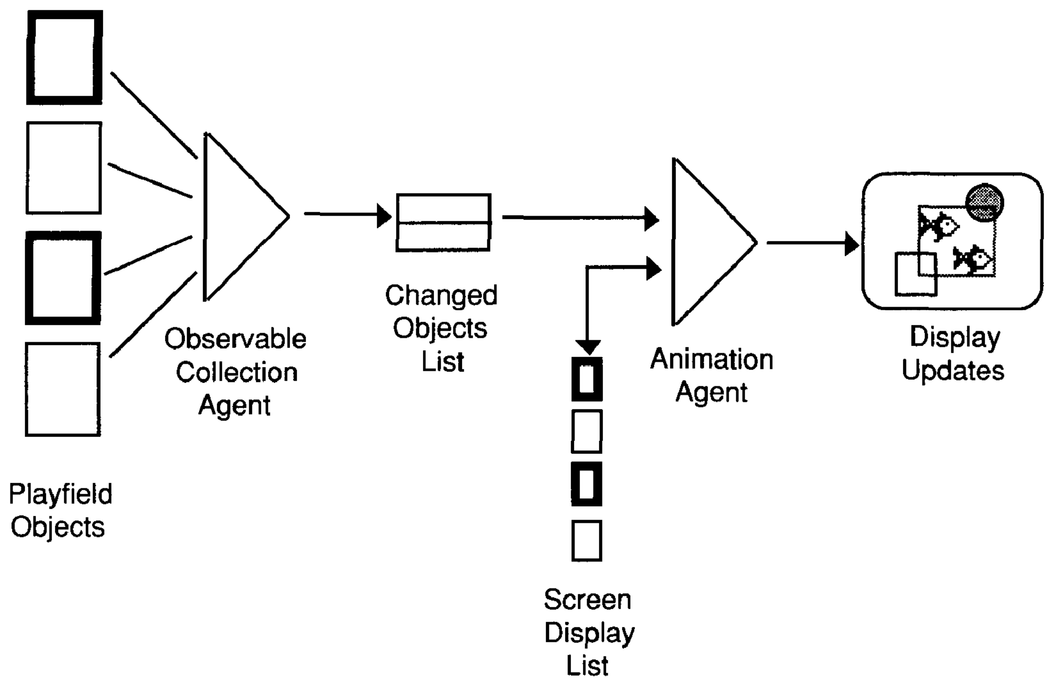


Figure 7: Animation Process

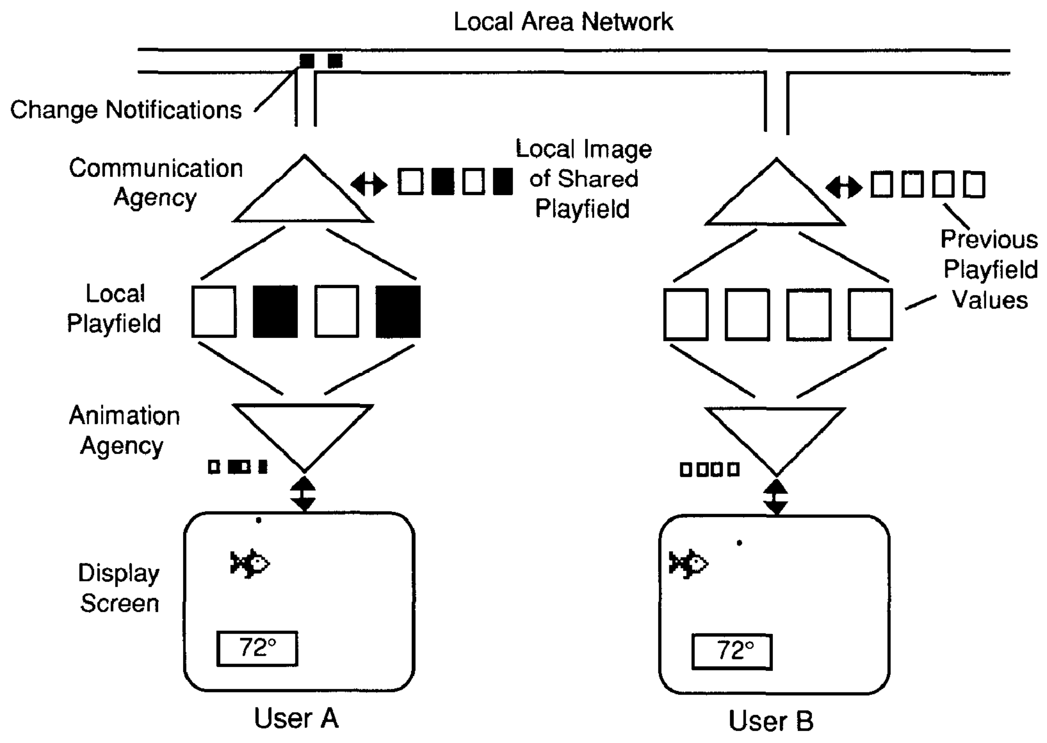


Figure 8: Telecommunications Process

Figure 8 shows how the telecommunications mechanism informs observers at remote stations about the motion of the fish and bubble objects.

History

Work on Playground began during the summer of 1986. During the past three years, three distinct versions of the program have been created and tested.

Playground 1 is coded in Lightspeed C for the Macintosh. The Playground 1 programming language used a functional notation similar to LISP but with fewer parentheses. Program statements were tokenized, then interpreted using recursive descent. Playground 1 achieved high animation and telecommunications performance.

Playground 2 is implemented in Digitalk Smalltalk/V [Digitalk88] running on the Macintosh II series of computers. The high interpretation overhead of Smalltalk forced us to emphasize language experiments at the expense of animation and communication. The first language tried was a modified SELF syntax [Ungar]. A SELF style delegation scheme was also tried and dropped in favor of a direct copying scheme.

Playground 3 is also implemented in Smalltalk/V and uses the operating environment created for Playground 2. The language was changed to the more natural syntax previously described in this paper.

The following table shows the same statement written in the three versions of Playground:

```
Version 1:  mousedown: moveto (shot,mousex,150).
Version 2:  notice: MouseDown.shot moveTo: (MouseX @ 150).
Version 3:  On mouse click move shot to MouseX @ 150.
```

We have found children to be most demanding and stimulating test subjects. In early testing with a more artificial concrete syntax, some children were still able to construct simulations of surprising complexity.

School Testing

Playground has undergone testing using groups of children at the Open School. Generally we do exploratory testing using small groups of 5 to 8 children first, gearing up for larger scale tests involving a class of 30 to 60 children taught by their regular teacher.

The Playground 2 version of the language was tested using a group of 60 fourth and fifth graders during the spring of 1989. The children were taught in two groups of 30. Each group met twice a week for 4 weeks. Each session lasted 1 hour. Each session opened with 20 or 30 minutes of demonstration and discussion, followed by a “lab” period where pairs of children completed the assigned programming exercises.

The material was presented in the following sequence:

• How to run and quit the Playground program. Use of user interface to create and modify graphical objects. How to name objects and give them rules. The coordinate system and how to make an object move by writing a simple agent. How to run and freeze the Playfield.

• Create random motion of objects by changing their speed and heading. Construct agents that display the internal states of other objects. Change an object's costume. Assign properties to objects. Simple counting.

• Construct agents enabling a “Fish” object to detect food and move toward it when hunger has grown high enough. Detect when fish is over food and eat it, resetting a “hunger” counter. Use of “clone” command to create multiple instances.

• Construct “Plankton” and “Plant” objects and have grow, reproduce, and die. Enhance fish to follow a life cycle as well. Count birth and death statistics.

Unfortunately, the spring experiment was curtailed due to a system-wide teachers strike. If the experiment had not been halted, we would also have covered the following:

- Add a "Shark" predator to eat the fish. Add agents to the fish to notice sharks and evade them. Have fish weigh hunger against fear. Study the predator/prey balance and population ecology.

- Construct other simulations such as video games and animated stories using Playground's features.

Fortunately, the strike was settled in time to be able to debrief the children and teachers approximately 3 weeks after the experiment was halted.

The students remembered quite a lot about Playground, even after having been away from it for 3 weeks during the strike and its aftermath. The children generally enjoyed using Playground, and most succeeded in accomplishing their assigned tasks. They were naturally annoyed at the bugs, unnatural syntax conventions, the relatively low speed of the interpreted environment, and deficiencies in error handling and reporting.

One child astounded us by creating an elaborate aquarium that included two species of plankton, a whale, jellyfish, seaweed, rays, fish, and crabs, using some features that were not explained in the workbook or by the teacher.

The Open School students had a long list of suggestions on how to improve Playground. The following list is an excerpt from our notes:

Q: What improvements should we make to Playground?

A: Build in video games. Add flip book animation. Speech synthesis. Speed it up. Put in a sound menu with many sounds and the ability to add more. Proximity detector. FullPaint and HyperTalk painting tools. Mouths that move as objects talk. VCR-type control panel. Realistic movement. Grouping of objects. More commands. Directed animation (frog sticking out tongue). Path animation, More shapes, spelling checker, study box to remember mistakes. Speech recognition, clairvoyant typing, color mixing, get rid of typing coordinates. Better command keys for run and stop (Enter and Space were suggested). Rotation of objects. Make screen bigger. Linked Playfields like HyperCard's.

Q: What sorts of things would you like to be able to do with Playground?

A: Social Studies, Sports, Race Track, Gymnasium, Cartoons, Science Fiction, Olympics, Westerns, Murder stories, Chemistry, Treasure Hunt, Trigonometry, Games, Music, Comics, Car crash, Sound Effects, Commercials, RoboCop, Spiderman, record what you are doing. Make a movie. Give characters different voices. Make a playground, hide in holes, talk to it (using speech recognition), a soap opera, space station. Soccer. Buildings. Make faces.

Conclusion

Our experience with Playground has encouraged us to explore several areas for making the system more accessible to children, including expanding or altering the user interface.

Interface

Comic books are well known for their popular appeal and offer a number of fruitful user interface ideas. For example, a sequence of operations can be expressed as a succession of panels. In addition, we could adopt a number of stylistic conventions for incorporating textual descriptions along with graphics. For example, the user could open a text editing balloon attached to a given object, and edit the text associated with it. The comic panel shown in figure 9 hints at what we are after.

Another significant problem we face is enabling children to design pleasing animal forms with engaging modes of movement. One promising approach is guided evolution, pioneered by Dawkins [Dawkins86], in which a constructed genome controls the creation of form, the genome is randomly mutated in several ways, the user selecting among them.

Other User Communities

We hope to eventually create a general purpose language for personal computer users. We need to explore ways of applying the Play-

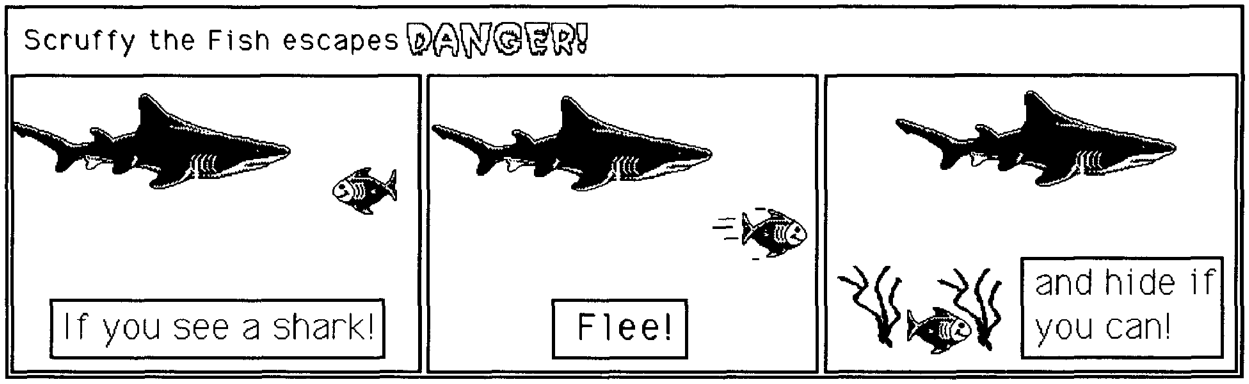


Figure 9: DynaComicBook panel

ground programming style to desktop programming problems.

Implementation

We are convinced that the pulling-style control structure has significant advantages over message sending. We have yet to implement a version of pulling that is efficient enough to be the basis of all computation. We are exploring techniques used in artificial intelligence for dependency management, hoping to gain enough performance for our next round of experiments.

Finally

We envision a system in which a group of children sit around a large, central screen showing the composite view of the Vivarium, rendered in full color in three dimensions. Each child has an individual screen on which to view and modify the shared world.

Through three-dimensional input devices with feedback, they design the form and behavior of a group of animals and are able to cooperate in building complicated individuals with sophisticated group behavior.

Perhaps the Central Intelligence Agency could use Playground to build a comprehensive simulation of the Soviet railway system that puts Lionel to shame. In any case, we look forward to being astounded with what the children of the world do with our system.

Acknowledgments

Alan Kay and Ann Marion deserve credit as codesigners of Playground, along with Kent Beck and Scott Wallace.Thanks are due to Mike Travers for suggesting the gated agent rule approach. George Bosworth contributed insights and code, Ted Kaehler, Steve DeWitt, and members of our illustrious advisory board have made their contributions to Playground design. Erfert Fenton helped edit this paper. David Mintz and B.J. Allen taught P to our kids, who themselves deserve honor for their pioneering spirit.

References

[Becker75] Joseph D. Becker, *The Phrasal Lexicon*, Bolt, Beranek, and Newman Report No. 3081, June 1975 (amusing and brilliant)

[Borning86] Alan H. Borning, "Classes versus Prototypes in Object Oriented Languages," *Proceedings of the ACM/IEEE Fall Joint Computer Conference*, November, 1986.

[Dawkins86] Richard Dawkins, *The Blind Watchmaker*, W. W. Norton & Company, 1986.

[Digitalk88] Digitalk, *Smalltalk/V Mac Tutorial and Programming Handbook*, Los Angeles, 1988.

[Fauconnier85] Giles Fauconnier, *Mental Spaces*, MIT Press, 1985.

[Gardin89] Francesco Gardin and Bernard Meltzer, "Analogical Representations of Naive Physics," Artificial Intelligence 38(1989) 139-159.

[Grier84] James W. Grier, *Biology of Animal Behavior*, Times Mirror/Mosby College Publishing, St. Louis, 1984.

[Ingalls88] Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph, Ken Doyle, "Fabrik - A Visual Programming Environment," *OOPSLA 88 Proceedings, San Diego*, pg. 176-190, 1988.

[Lakoff87] George Lakoff, *Women, Fire, and Dangerous Things*, The University of Chicago Press, Chicago, 1987.

[Minsky85] Marvin Minsky, *The Society of Mind,* Simon and Schuster, New York, 1985.

[Papert80] Seymour Papert, *Mindstorms: children, computers, and powerful ideas.* Basic Books, New York, 1980.

[Smith86] Randall B. Smith, "The Alternate Reality Kit: An Animated Environment of Creating Interactive Simulations," *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages*, Dallas, TX, June 1986, pg. 99-106

[Tinbergen51] Niko Tinbergen, *The Study of Instinct*, Oxford University Press, 1951

[Travers88] Mike Travers *Agar: An Animal Construction Kit*, Unpublished masters thesis, M.I.T. media lab, 1988.
(contact us for a copy)

[Ungar87] David Ungar and Randall B. Smith; "Self: The Power of Simplicity," *OOPSLA '87 Conference Proceedings* pg. 227-242, 1987.

[Wilensky87] R. Wilensky, *Some Problems and Proposals for Knowledge Representation.* Computer Science Division. University of California - Berkeley, Report No. UCB/CSD 87/351.

Appendix: Abridged Playground 3 vocabulary

set NAME to VALUE — set value to a slot.
set curiosity to 30.

angle; set angle to VALUE — controls the direction an object will travel. Angles are in degrees following the compass rose.
set angle to 90. "gets player going east"

bearing from VALUE {to VALUE} — the degrees to the object or point given as an argument.
display bearing from 200 @ 100 to Sam.

bounds — gives the bounding rectangle.
The boundaries of an object can be referred to by phrases like these:

my top right corner
my center
the bottom right corner of VALUE and so on.

Each of the above can also be used in the set command, for example:
set my top right corner to the center of cactus.

clock — how many ticks of the Playground world have gone by.

clone yourself — make an exact copy of this object and give it an independent existence in this world.

change costume to VALUE — set the way an objects looks.
set costume to black rectangle

depth, set depth to VALUE — change depth coordinate to new value given.

display VALUE — change my costume to display the number or name given as an argument.

distance from OBJECT {to OBJECT} - calculate the distance from one object to another.

NAME from VALUE — returns the value of a property from another object.

go forward VALUE steps — move current object forward the number of units requested.

go to POINT — move object towards a specific point at the current speed.
go to 100 @ 150.

grow by VALUE — change size by factor given.

if VALUE then STATEMENT — execute STATEMENT only when VALUE is true.
if bozo < 12 then display 'that"s a bozo no no!'.

make sound from file STRING — trigger playing sound with name given.
make sound from file 'monkey'. The quote marks are required.

move by XYAMOUNT — move the object by an amount on both x and y axis.
move by -20 @ -2. (moves 20 to the left, 2 up)

move to LOCATION — move to the location given.
move to 100 @ 200

nearest object with property NAME — returns nearest object which has the property requested.
go to (nearest object with property green) center.

notice CONDITION — introduces the condition part of an agent rule.

number — retrieve the value of the costume as a number.
display number * 2. "double each tick of the world"

over OBJECT — returns true when one object overlaps another.
over somebody with property NAME — sets true if one object overlaps another object which has a certain property.
notice over somebody with property food.

VALUE random — returns a random number between 0 and 1 less than VALUE.

remove yourself — removes this object from the world forever.

set result to VALUE — set result cell to value given.
return VALUE — return value as result for this agent.

my result — return result for this agent.
result for NAME — return result for another agent in this object.
result for NAME {in VALUE} — return result for another object.

speed,set speed to VALUE — sets the speed that an object should move in pixels per second.

wait VALUE ticks — causes THIS AGENT to wait the requested time interval before continuing execution.

who i am over {with property NAME} — returns a pointer to whatever object I am over that has the property requested, if any.