Computer Games for Kids, by Kids

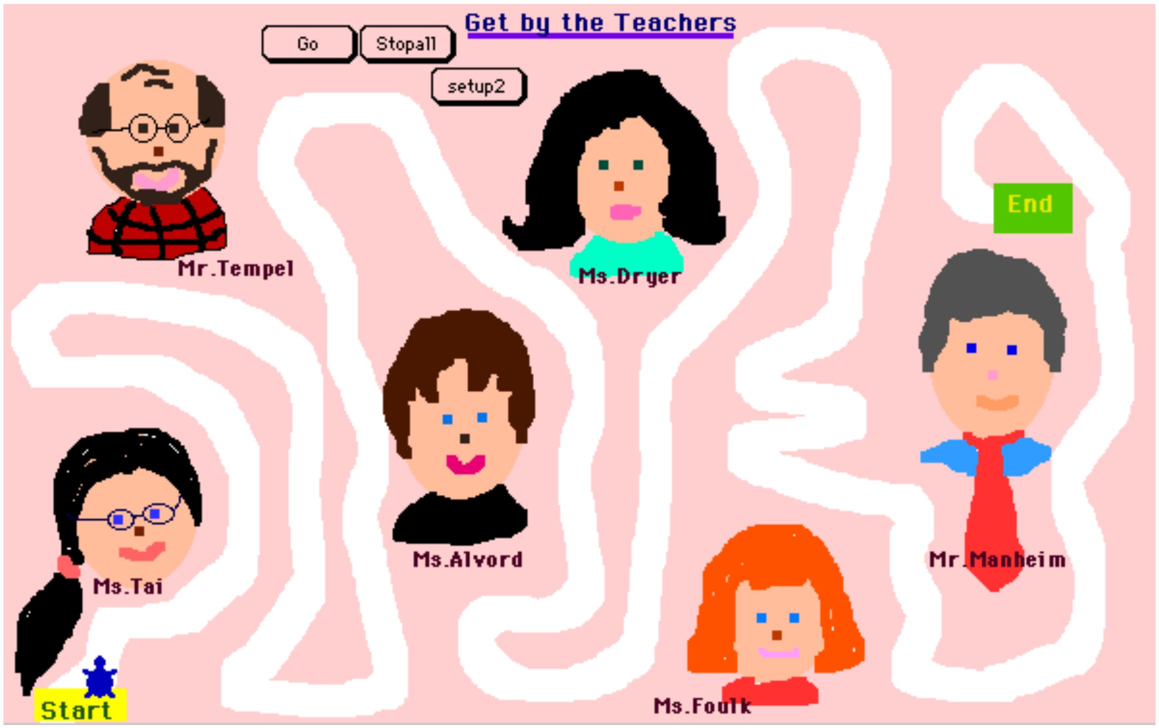
by Michael Tempel[1] and Hope Chafiian[2]

Kate was working on a computer game. Dogs were running, birds were flying, messages were popping up, questions were being asked, and music was playing. She wasn't just playing the game. She wrote it. It was January and she wasn't scheduled to take computer until April, but she had learned what she needed to know from friends who had already taken the course.

A group of students corner their teacher in the stairwell to tell her about a bug in their program. Another wants to know why the timer isn't working in her program.

Four students are grouped around a computer watching intently as a fifth tries to maneuver a fast-moving turtle through a winding maze. His fingers move rapidly around the arrow keys as he tries to avoid going into the wall. He fails and is sent back to the beginning of the maze. One boy in the group is quite pleased. He is the author of the program and his creation has remained undefeated.

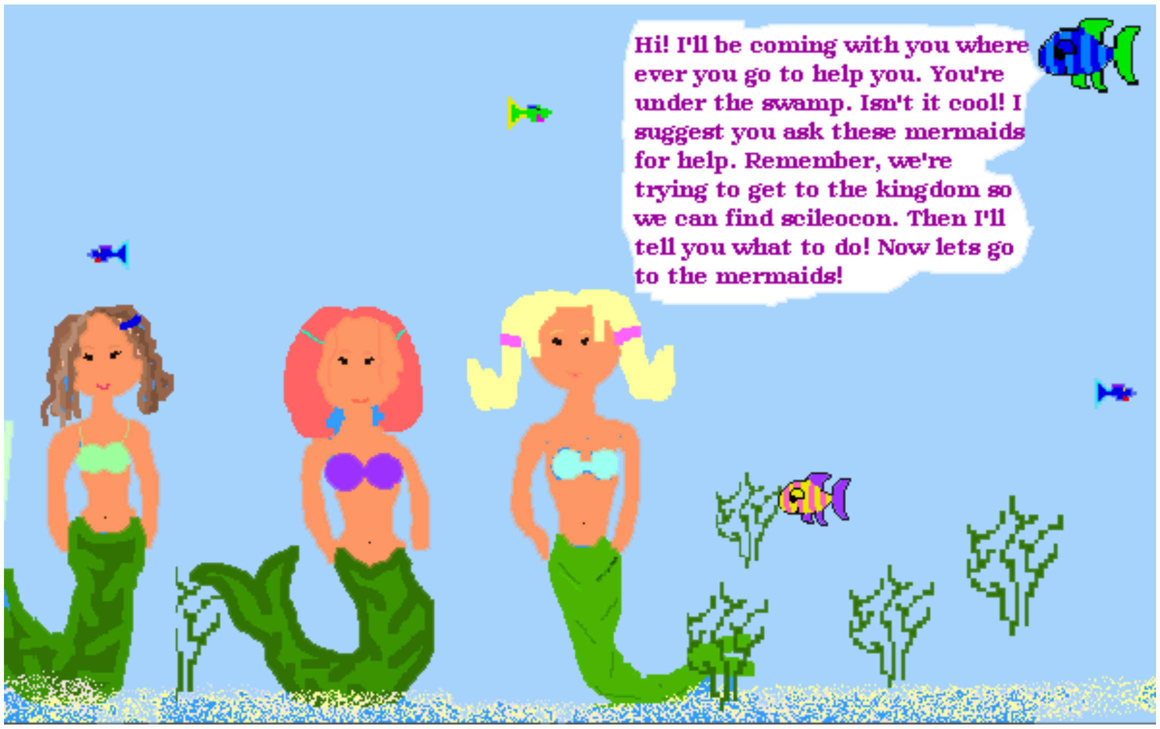
These students are thinking a great deal about what they are doing. They are involved. Educators have long taken advantage of children's passion for games, and especially for computer games. The usual approach is to overlay some educational content onto a familiar game format. To reach the next level you must answer a question about history or calculate a sum.



Try to maneuver around the teachers in Elizabeth's Maze Game

We don't ask our students to design games that focus on a school subject. They choose what the games are about so the objective is more likely be to conquering space aliens or getting a date than solving a math problem or getting to the West Coast.

We follow in the tradition of Idit Harel[3] and Yasmin Kafai[4] of the MIT Media Lab, recognizing that more profound learning comes from designing and building the games than from being on the receiving end.


Hi! I'll be coming with you where ever you go to help you. You're under the swamp. Isn't it cool! I suggest you ask these mermaids for help. Remember, we're trying to get to the kingdom so we can find scileocon. Then I'll tell you what to do! Now lets go to the mermaids!

Ask the mermaids for help in getting around Stephanie's adventure game.

The Settings

We have been working with students on computer game projects for several years, at The Spence School, a K-12 independent school for girls, and at Computer School II, an alternative public middle school, both in New York City. We have also shared our work with other teachers in workshops on game programming and as part of the annual Logo Summer Institutes sponsored by the Logo Foundation.

By the fifth grade, Spence students are old pros at Logo. They have been using MicroWorlds since kindergarten and have a variety of projects saved in their accounts on the school's network. Spence students know turtle geometry well and have used it to explore concepts of angles, area, perimeter, polygons and fractions. They have generated probability outcomes and have developed games around probability. They have programmed geometric faces, animations and turtle races. They have used MicroWorlds for creating multimedia reports, importing graphics from the Internet, digital cameras and scanners.

At Spence the fifth grade computer class, which meets Tuesday and Wednesday afternoons in double periods for one third of the year is one of a series of classes, including dance, drama, and writing workshop, that provide varied environments and tools to foster students' self-expression.

Computer School II opened in September 1996 with 45 students in 6th and 7th grades. It expanded the following year to include an 8th grade and approached its target enrollment of 90 to 95 students. We began with 25 networked computers in one room. By the middle of the 1997 - 1998 school year there were 55 computers on the network with at least four machines in each classroom in addition to those in the computer room. Internet access was limited to one computer using an unreliable dial-up connection.

All students and staff members had accounts on the file server, and could access their work from anywhere in the school. This server also allowed us to set up a “Public Folder” that everyone could look at and copy files from. This shared area was important for the game project because we could put starters and samples there to introduce new ideas and techniques. The Public Folder also provided an exhibit area for finished work.

Students had four computer periods per week. The computer room was also full during lunch periods and many students stayed for two hours or more after school. Computers were used in conjunction with most subjects, but the dominant activity during computer classes was the game project.

At Spence and at Computer School II the computer classes develop into active collaborative design studios. Most students are quite willing to help each other, although some form groups that harbor secrets. Since the students are motivated to improve their games, there is a drive to learn new skills and acquire information. It’s a pleasure to have them gobble up new ideas and techniques. Whole class lessons are rarely needed or appropriate. We often teach something to one student knowing that it will be all over the class in short order. Or, we place a sample project or starter in the Public Folder on the network and students take it as they need it.

What We Want to Achieve

What are our students learning? There are basic language and math skills that are practiced and improved while building games. Instructions have to be clear. Calculations of distance and angle are needed to lay out a game and move characters around. They are learning to plan and organize a large project and to appreciate another’s point of view, that of the person playing the game. They are learning programming and the important skill of debugging.

As teachers we have our criteria for judging our students’ work. But what drives them most to high achievement is the judgment of their peers. Games should be complicated and difficult, but fair and possible. It should be clear what the goal is and what you have to do to maneuver your characters. Games should be clever, tricky, and funny. They should be aesthetically pleasing. And they should be fun.

Getting Started

At Spence the game project began with Madlibs. At Computer School II we started with Mazes. In short order we introduced other types of games: puzzles, adventure games, and board games. Many projects were mixtures of these kinds of games.

Madlibs at Spence

Our work has been heavily influenced by Paul Goldenberg's and Wally Feurzeig’s book *Exploring Language with Logo*[5] . The authors examine natural language structure through Logo programming. In the past we have tried some of the projects explored by the authors, including Gossip, a program that randomly generates noun phrases and verb phrases to create sentences; Plural Nouns, which takes a singular noun as input and reports its plural; Conversations, and Madlibs.

We begin the fifth grade class with a Madlibs starter . By fifth grade the girls at Spence have had plenty of Logo experience, but have done little with the English language itself in computer class. Madlibs is a good project choice It is an interactive game familiar to kids, usually has a funny outcome, is a rich English grammar activity and provides a great introduction to the use of variables and to Logo grammar.

Many of us played Madlibs as children. The game consists of a story that has some missing words. For each blank space there is the name of a part of speech. One player reads off these parts of speech and the other player, who does not see the story, supplies a word that fits the category. For example one might say “table” for a noun and “quickly’ for an adverb not knowing the context that these words will end up in. When all the blanks are filled in the player with the pad reads the story with the other player’s words plugged in.

With Madlibs pad and pencil in hand we were occupied for hours. However, we don't ever remember making our own Madlibs. It’s not that you can't do it without a computer - but it never occurred to us to try. An Internet search will find many sites where people can play Madlibs by typing in the parts of speech. There are even some sites that let you "create your own" Madlibs story. But in this context, creating your own requires no programming skills. The website takes care of that for you. Creating Madlibs with Logo puts the students in total control as both programmers and players and enhances the activity in so many ways. The computer environment fosters a collaborative atmosphere and Logo provides immediate feedback and helpful error messages. Students realize the need for testing programs over and over again, debugging as they go along, offering and accepting constructive criticism from their peers.

Madlibs is a fun activity that reinforces one's understanding of parts of speech, pluralization and conjugation of verbs. By the fifth grade, the parts of speech have already been taught, but their labels - adjective, noun, verb, and adverb - are not yet internalized. Many kids have to talk it out loud: What kind of word is “happy?” What's an adverb? The students do have a good sense that words play different roles in sentences, but they can’t always express rules of grammar in a formal way. This is where programming is crucial. When students test their Madlibs they are testing them for programming bugs as well as for English language bugs.

The Starter

Students begin with the Madlib starter. It is simple and short. It introduces **question**, the concept of creating variables, and putting together words and sentences.

```
to madlibs
announce [Hi there.]
question [ please type in a noun.]
name answer "noun1
question [type in an adjective.]
name answer "adj1
question [ please type in a plural noun.]
name answer "noun2
pr (se [The] :noun1 [is] (word :adj1 ".) [ I really like ]
(word :noun2 ".))
end
```

We ask students to play the starter a few times, each time typing in different responses, until they get the hang of it. We look at the procedures and discuss the role of variables. We suggest that the students first write an entire story. Then we give them some tips for turning it into a Madlib:

1. Use brackets to surround the text that should remain part the story.
2. Substitute variable names such as **noun1** or **adj1** for the words that will be entered by the person playing the game.
3. Write the prompts using **question**, for example **question [please type in a noun.]**
4. Use **name answer** to set the value of the variable to the words that the player types in, for example **name answer "noun1.**

Students tackle the Logo bugs first. They must rely on their inferencing skills every time they get an error message. The most common types of errors that arise include missing brackets and colons, misplaced parentheses, and putting a space before a colon or after quotes.

They know enough to make their Madlibs work. Using our starters and other students' programs, even if there is only a partial understanding of the code, supports the learning of Logo ideas. In time, with lots of exploration and project building, students will deepen their understanding of how Logo works.

Once the Logo code is bug free, students may think they are finished. However, the hard work begins when they engage with language at a deeper level. The ultimate goal of a successful Madlib is to end up with a funny story. There is a fine line between a silly story, and a story that just doesn't make sense because of the misuse of adjectives, nouns, verbs or adverbs.

Debugging English may be more difficult than debugging Logo. The computer only reports Logo errors, not English errors. It takes some sophistication to zero in on the troublesome places: when to call for a plural noun rather than a singular noun or what verb tense should be used. Here's an example of a Madlib that works.

```
to madlibs
question [Hello, my name is Olivia, what's your name?]
announce [That is a very pretty name.]
question [Would you like to play my game now?]
announce [Okay.]
question [Please type in a food.]
name answer "food
question [Please type in a color.]
name answer "color1
question [Please type in another color.]
name answer "color2
question [ Are you starting to get the hang of it?]
announce [Good!]
question [Please type in a shape.]
name answer "shape
question [Please type in a noun.]
name answer "noun
announce [ We're all done, you did great!]
announce [Goodbye!]
pr (se [There once was a cow who lived on a farm, her name
was Spot. Spot liked to eat] (word :food ".) [Spot was
definitely not an ordinary cow, she was] :color1 [and] (word
:color2 ",) [her head was in the shape of a] (word :shape ",)
[she slept on a] (word :noun ",) [and took baths in a
bathtub. Spot was one of a kind.])
end
```

This next example was generated by a Madlib that was still in the process of being developed. The Logo program works, but the Madlib isn't quite free of English bugs. The words in italics are the responses typed in by the user.

```
REPORT
Dear Mr. and Mrs. Chafiian,
Your daughter has been a big problem. She has cheated on all
her shoe and is mean to all her desk . Half of the grade has
complained about Hope . On her record it says that she was
suspended from 45 grade for biting someone's knee . Also her
teachers have some bad reports about her. She has rudely
drink back to them. Please get your daughter under control.
If you can't you are pretty parents.
unsincerely,
Mrs. smith principal
```

By programming Madlibs in Logo students are working with the English and Logo languages in parallel. They are simultaneously focusing on Logo syntax and English syntax. The fact that the students are learning about the rules of Logo grammar by doing English grammar, suggests that they are engaged in abstract parallel thinking. We have observed over the years that the greater the understanding of English grammar, the more clever and witty the Madlibs.

Mazes at Computer School II

Students at Computer School II come from many different elementary schools throughout Community School District 3, which covers Manhattan’s West Side and south central Harlem. A few are from public schools in other districts, independent schools, or parochial schools. They come with a wide variety of school computer experiences. About half the students have computers at home. Hardly anyone knows Logo.

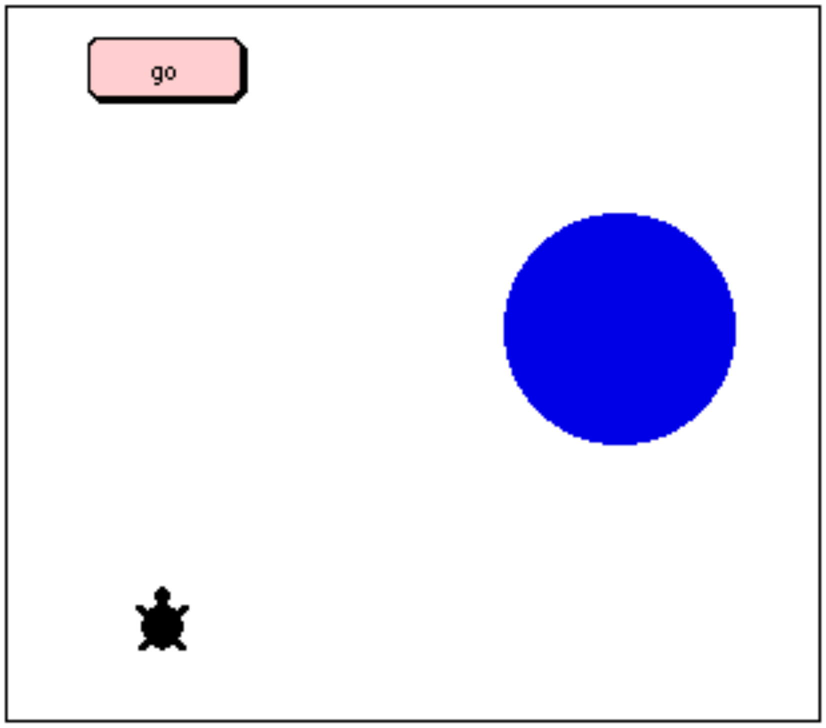
Prior to beginning the game project about six weeks into the first school year at Computer School II we spent some time gaining familiarity with MicroWorlds. The project “Creating a Dynamic Story” from the *MicroWorlds Projects Book* provided an introduction. We followed with some turtle geometry, which was related to a unit in the math curriculum on polygons and mosaics. With this background the students had enough collective knowledge and skill to begin their games.

Again, we used the *MicroWorlds Project Book*. The chapter “An A-Mazing Project” was a good starting point with its suggestions about how to create a maze and direct the turtle through it. But the kids did not like changing the turtle’s heading by clicking buttons on the screen. They wanted to use the arrow keys. We put a project in the Public Folder that had the information needed to do this. Students could copy it and incorporate the procedures and techniques into their own projects.

Additional starters were put in the Public Folder as needed to illustrate various aspects of game programming. When more than a few students were asking the same question it was probably time for a new starter. Information spreads from student to student anyway, but putting something on the network and calling attention to it serves two purposes. First, it is a way of focusing on an idea or technique that we consider to be important. Second, it insures that everyone is aware of it. Informal communication would not necessarily reach everyone.

The sequence of starters below is based on the ones we used at Computer School II and in the many workshops we have done with teachers. They are designed to get things going. Once the students are actively working on games, they move in many different directions. Increasingly, their own work takes the place of these starters as examples and material for others to follow.

This first starter just shows how to move the turtle with the arrow keys to arrive at a target.[6]



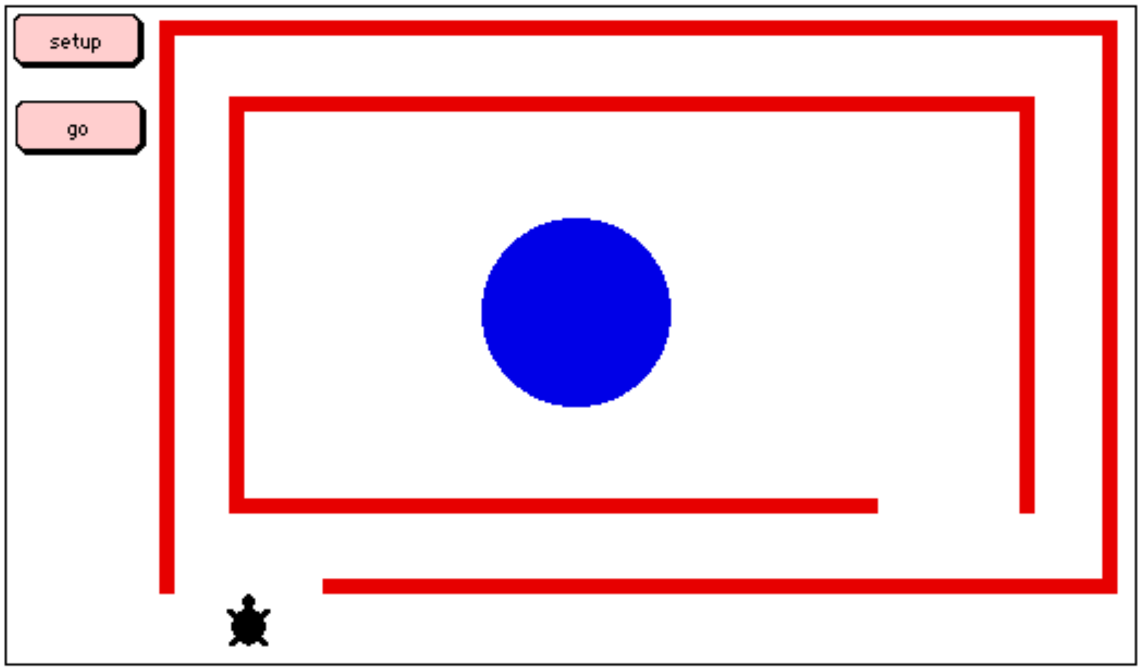
The procedures page looks like this:

```
to go
t1,
 clickon
 forever [direct readchar]
end

to direct :key
if (ascii :key) = 28 [ seth 270]
if (ascii :key) = 29 [ seth 90]
if (ascii :key) = 30 [ seth 0]
if (ascii :key) = 31 [ seth 180]
if :key = "s [stopall]
end
```

The turtle is programmed to go **forward 1** many times. The color blue is programmed to run the instruction **announce [you win!] stopall** when the turtle touches a blue part of the background.

This second starter is the same as the first with the addition of a maze and a setup procedure.

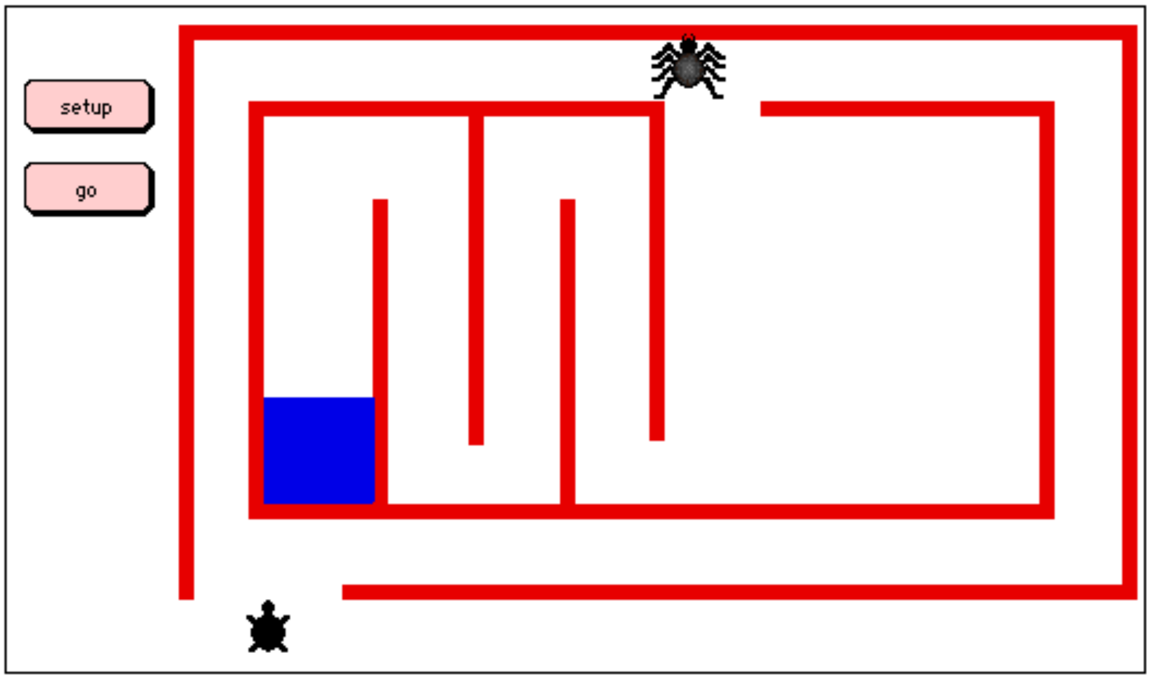


The color red is programmed to turn the turtle **right 180** when it bumps into a wall. This is the **setup** procedure:

```
to setup
setpos [-150 -130]
seth 0
end
```

It is used to place the turtle at an appropriate starting position and heading for the game to begin. The way to determine the proper position is to use the mouse to drag the turtle to where you want it to start and then type **show pos** in the command center. The pair of numbers reported can then be copied into **setup** on the procedures page. The heading can be determined by typing **show heading**, but this is often unnecessary because the students recognize headings of 0, 90, 180, and 270 degrees.

The third starter adds a moving obstacle: a turtle in the shape of a spider.



This spider turtle is also programmed to go **forward 1** and its heading is set to 90. This is done by typing the command **seth 90** in the command center. Once it is moving the spider turtle will switch headings between 90 and 270 each time it bumps into the red walls because red is programmed to cause turtles to turn **right 180**.

Adding the spider also requires changes to the **go** procedure.

```
to go
talkto [t1 t2]
 clickon
forever [direct readchar]
when [touching? "t1 "t2] [setup]
end
```

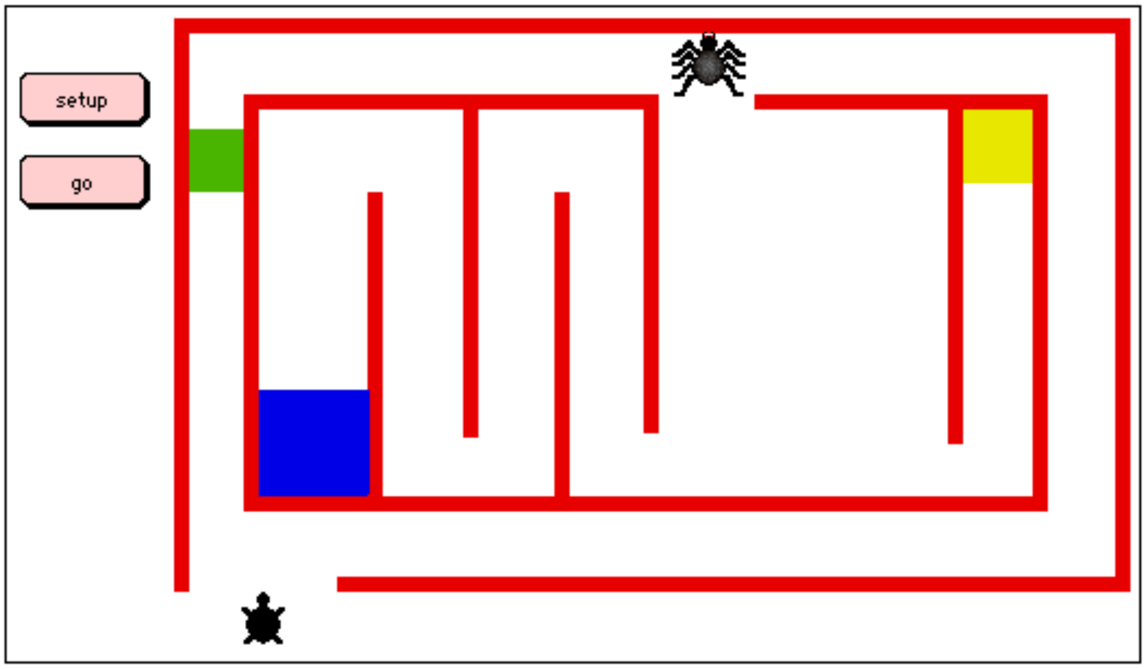
Both turtles must now be told to **clickon** and the collision detection is established with the **when** command. Setup is also changed so that it only talks to the original turtle and not the spider as well.

```
to setup
t1, setpos [-140 -130]
seth 0
end
```

Without this change both turtles would be sent to the starting gate when there was a collision.

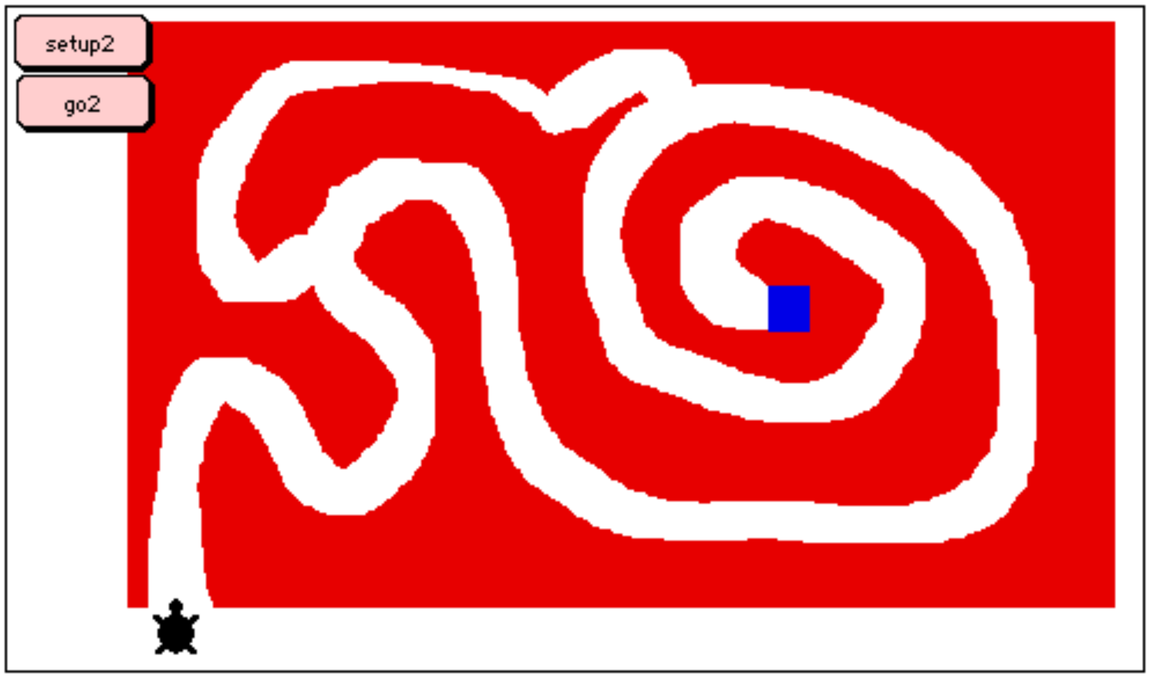
Most students did not use moving turtles in their mazes, but collision detection was used in other kinds of games. Pong, Air Hockey, and War Zone on the VideoGameWorks website[7] are examples.

The fourth starter adds two more blocks of color in the maze. One of them sends you back to the start while the other jumps you closer to the blue goal.



Green, on the left, is programmed with **setup** while yellow, on the right, is programmed to **setpos[-67 -37]**. Each page can have as many as 16 programmed colors. This technique was used extensively in students' games. Talia's Maze, and Loren's Maze on the VideoGameWorks website are good examples of this.

The fifth starter has a second maze on page2.



The blue target on **page1** now has the instruction **announce [you win!] page2 stopall.** On **page2** there is a **setup2** button and a **go2** button. The procedures they call are:

```
to setup2
setpos [-190 -130]
seth 0
end

to go2
clickon
forever [direct readchar]
end
```

A common error is to program the target color blue with

```
announce [you win!] stopall page2
```

The programmer is thinking, “I want to announce the win, then stop everything on the current page, then go to the next page for another level of the game.” Unfortunately, **stopall** means just that. It stops everything, including the process that started when the turtle ran into the color blue, so the command **page2** is never reached. By putting **stopall** after **page2** you arrive at **page2** ready to click **setup2** and then **go2**.

There are other ways of ending the action on one page and then moving to another, but these are generally more complicated. You have to be aware of which processes are active and stop each one. **Maze5** could be modified to have the color blue programmed to run the procedure **endgame1**, which would look like this:

```
to endgame1
announce [you win!]
ask [t1 t2] [clickoff]
cancel [direct readchar]
cancel [touching? "t1 "t2]
page2
setup2
go2
end
```

After the winning announcement the instructions in both turtles are turned off with the command **clickoff**, the two active processes are **cancel**led and we go to **page2**. Then, **setup2** and **go2** are run causing the action to start immediately when the page is opened.

A Closer Look at the Students’ Games

The students elaborated upon the techniques and ideas in the starters and added many new features. It was common for games to have many levels. Sometimes these levels were organized into a story. The goal on the first level of Elizabeth’s CS II Maze is to get to school. Having accomplished that, you then have to get by the teachers, make it through the lunch room, recess, and back to class. Finally, if you make it home your reward is to have a good weekend.

Most of the mazes set up winding curving paths rather that the simple rectilinear layout shown in the starters. This makes them more difficult since the turtle’s heading must be either 0, 90, 180, or 270 degrees.

In the starters bumping into a wall causes the turtle to turn **right 180** and go in the opposite direction. The students usually program a much harsher penalty: jumping back to the beginning of the maze, or having the game end. It was also common to use **announce** to denounce the player for clumsiness, or to record the message and program the name of the recording into the wall color so that it played when a collision occurred. In Loren's maze the character explodes when he hits the wall. The explosion is a series of 10 shapes that are displayed in sequence. These shapes were imported into MicroWorlds and found their way into many other games in addition to this maze.

Loren's maze has another trick that takes advantage of the fact that the turtle "wraps," that is, when it goes off the right side of the screen it reenters on the left side and when it goes off the top it comes back on the bottom. The path of Loren's maze repeatedly exits one side and continues on the opposite side. Not only that, the color of the path changes. It's rather difficult to follow.

Some students introduced speed control. In Crystal's Maze there is a slider named **speed** that may be set to a number 0 to 50. The turtle instruction is **forward speed** instead of **forward 1**. **Speed** reports the setting of the slider to **forward**.

There's More

The games we have posted on the Web[7] are representative of many others done at Spence and Computer School II. These are better than most in terms of being more fully debugged, checked for spelling and grammar, and having instructions. These are standards that we as teachers set for public posting of the games. There are many excellent games that were shared locally but are not on the Web because of technical limitations. Sounds are difficult to transfer from Macintosh to Windows and some of the projects were so large that the download time for viewing with the MicroWorlds Web Player would be very long.

There are also many "works in progress" that are quite good, but not ready for prime time. Sometimes students lost interest in a game before finishing or ran into technical problems that didn't get solved. It was common to keep pushing the limits of what the programming environment could handle until things stopped working smoothly. At that juncture we encouraged backing up to a point where the game did work well, tying up loose ends, and getting the game into a presentable form. Sometimes this happened, sometimes not. For every game that was placed in the Public Folder or on the Web there were many more that were left at varying stages of development. In an exploratory environment this is to be expected. It is important to have finished products to show, but the learning is mostly in the development process.

[1]Michael Tempel is President of the Logo Foundation. He may be contacted at:

Logo Foundation
250 West 85th Street
New York NY 10024
USA
phone 212 579 8028
fax 212 579 8013
michaelt@media.mit.edu
http://el.media.mit.edu/logo-foundation/
http://www.logofoundation.org

[2]Hope Chafiian is Director of Technology and Curriculum at The Spence School. She May be contacted at:

The Spence School
22 East 91st Street
New York NY 10128
USA
phone 212 289 5940
Hope_Chafiian@spenceschool.org

[3]Harel, Idit, *Children Designers,* Ablex, Norwood, NJ, 1991

[4]Kafai, Yasmin, *Minds in Play,* Lawrence Erlbaum Associates, Mahwah, NJ, 1995

[5] Goldenberg, E. Paul, and Feurzeig, Wallace , *Exploring Langauge with Logo*, MIT Press, Cambridge, MA, 1987

[6]The ascii numbers for the arrow keys are not the same on all computers. On a Macintosh they are 28, 29, 30, and 31. On a PC the corresponding numbers are 37, 39, 38, and 40.

There is a minor problem with **readchar** that is easily overcome. If the cursor is active in the command center **readchar** will not read what you type. Clicking somewhere on the MicroWorlds page makes **readchar** responsive. In practice this is not a problem with the games because they are started by clicking a button on the page.

Students sometimes want to know the ascii numbers for other keyboard characters. This is usually unnecessary since any printable character may be used directly as in the last line in **direct** :

```
if :key = "s [stopall]
```

Still, for characters such as tab and space it is necessary to know the secret numbers, and some students are just curious. The trick to finding out is the instruction **show ascii readchar**. When you run this instruction, Logo waits for you to press a key and reports its ascii number in the command center. However, if you type the instruction in the command center you then have to click on the page to make **readchar** responsive. To avoid this we usually make a button with the instruction **show ascii readchar.**

[7] The VideoGameWorks Website is at http://el.www.media.mit.edu/logo-foundation/VGW/