

MicroWorlds EX™



Exploring with MicroWorlds **EX**
Projects



Exploring With MicroWorlds**EX** **Projects**

by Gary Stager and Susan Einhorn



© Logo Computer Systems Inc., 2003. All rights reserved.

No part of the document contained herein may be reproduced, stored in retrieval systems or transmitted, in any form or by any means, photocopying, electronic, mechanical, recording or otherwise, without the prior approval from Logo Computer Systems Inc.

Legal deposit, 1st semester 2003

ISBN 2-89371-528-1

Printed 6-03



MicroWorlds EX is a trademark and **LCSI**® is a registered trademark of Logo Computer Systems Inc.

Table of Contents

Overview	3
Section 1: Models and Simulations	5
Exploration 1 - Gas Atoms in a Jar	6
Exploration 2 - Environmental Chaos	13
Exploration 3 - Of Dice and Math.....	31
Section 2: Videogame Design	49
Game 1 - Snacman	51
Game 2 – Squish the Turtle: A Web-ready Game	67

Overview

The MicroWorlds EX Projects Book contains two main sections. The first contains several projects in which you can investigate ideas in math and science by creating on-screen simulations and explorations. The second section introduces you to the world of game-building. Although games are fun and, well, yes, games, they also involve a great deal of logic and problem-solving. You'll find that building games is even better than playing them.

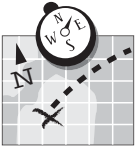
You can do these projects in any order. All the prerequisite skills are introduced in the MicroWorlds EX tutorials, although a number of those skills are also reviewed in this book. Before starting this book you should be familiar with:

- MicroWorlds EX objects, such as text boxes and turtles - how to create them and set them up
- the Painting tools,
- the turtle's backpack,
- creating animation,
- defining procedures,
- messaging and broadcast,
- randomness (although you learn many more uses for random in this book),
- turtlesown properties, and conditionals.

Simulations and games have many characteristics in common. Randomness plays a role in almost all the projects included in this book. See if you can find other ideas and/or techniques that these two types of projects have in common.

Remember, these are starting points for your own personal projects – extend and/or change them as much as you want. If there's a technique you've forgotten how to do or that you want to learn, refer to the Techniques panel in the MicroWorlds EX Command Center.

When you work through the book, you'll notice the following:



This icon indicates the description of the overall project plan.



This icon appears when there's a definition of a MicroWorlds EX primitive or an explanation of a specific programming technique.



This is a bug box. Explanations found here tell you what to do if something goes wrong or doesn't work as you expect.



This icon appears next to a list of ideas that are additional features you may want to add to your project.

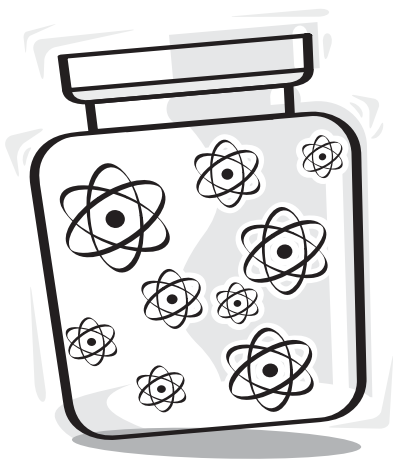
Sometimes the key combinations are different for Windows and Macintosh versions. In these cases, the Windows key combination appears first, the Macintosh, second; for example: Right-click/Ctrl-click.

Section 1

Models and Simulations

Professionals in all walks of life use computers to create numerical or visual models of real-world phenomena. These models may be used to illustrate an argument, make sense of data or predict a result if variables were to change. The computer offers a low-cost, quick and risk-free (nothing should explode) way of modeling. The computer's ability to perform repetitive computational operations quickly, accurately and without tiring makes it a powerful tool in the sciences and social sciences. Models in which particular variables may be changed are known as simulations.

This section presents three examples of the types of models you can construct with MicroWorlds EX.

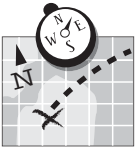


Exploration 1 -

Gas Atoms in a Jar

This project uses multiple turtles to represent atoms in a jar. The behavior of the simulation changes based on the number of atoms you place in the jar. Atoms move randomly and accelerate whenever they bounce into the walls of the jar or other atoms. The atom escapes this chaos when it hits the open mouth of the jar.

For the sake of simplicity, new atoms will be created and added to the jar by copying and pasting an existing atom (turtle). There are more complex ways in which we could build an interface allowing a user to click a button and have a new atom introduced into the experiment.



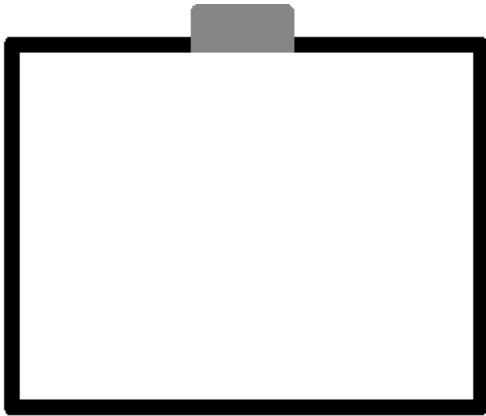
In this project you:

- Draw the jar
- Create atoms
- Set the atoms' rules
- Run the experiment

Getting Started

Start by creating a new project. Name the project and remember to save often.

Use the Painting tools to paint a jar that looks like the following. Use thick black lines for the walls of the jar and pink for the opening at the top of the jar.



Hatch a new turtle

- In the turtle's Shapes Tab, create a small ball shape and set the turtle to this shape. The color and size are up to you, but keep it simple.



The speed variable

In this experimental model, speed is one of the dependent variables. You need to have a way to set up the starting speed any time you run your model.

- Each atom should have a property called speed. To create this property, type the following in the Command Center:

```
turtlesown "speed
```

- Type the following in the Command Center:

```
setspeed 0.1
```

This sets the initial value for the current turtle-atom's speed.

Collisions

When the turtle-atom collides with the side of the jar or another turtle, it should bounce and accelerate a little. That means it needs a bounce procedure.

Since all the atoms will use the exact same procedure, you can write it in the project Procedures Tab.

```
to bounce  
bk 20 rt random 360  
setspeed speed + 0.1  
end
```



Speed reports the current (pre-bounce) value of speed.

This procedure causes the turtle to back up, turn randomly and increase its speed slightly. Note – use .1 as the increase in the speed so that the simulation doesn't come to a boil too quickly!

Now, give the turtle its rules. Open the turtle's backpack and click on the Rules Tab

- Type the following in the **OnClick** instruction field:

fd speed

Set it to Forever mode. This rule lets you start the atom by clicking on it

- Type the following in the **OnColor** instruction field for black:

bounce

Leave it set to Once mode. This rule tells the turtle to bounce and accelerate when it touches the wall's of the jar.

- Type the following in the **OnTouching** instruction field:

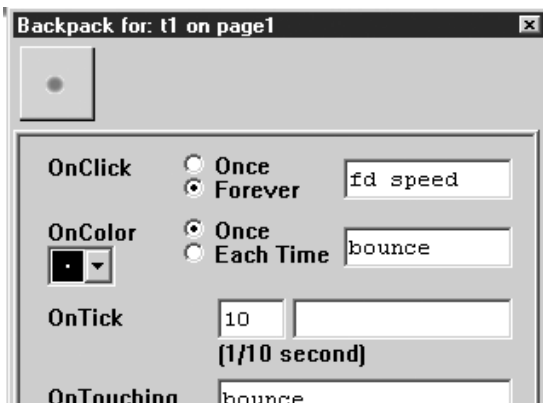
bounce

Leave it set to Once mode. This rule tells the turtle to bounce and accelerate when it collides with another turtle.

- While in this tab, type the following in the **OnColor** instruction field for pink:

clickoff

Leave it set to Once mode. This rule tells the turtle to stop when it touches the pink mouth of the jar.



A color that has a dot in its middle has instructions associated with it.

Testing the experiment

- Put the atom in the jar
- Click on the atom to set it in motion
- Observe its behavior

Does the model work the way you had planned? If yes, it's time to add more atoms.

Create more atoms

In the Command Center, type:

clone *turtle-name*

Use the name of your atom-turtle in place of *turtle-name*
(for example, **clone "t1"**).

To re-start the experiment, type the following in the Command Center:

everyone [clickon]

Reset your experiment automatically

In order to set your experiment back to its initial state again, it's a good idea to write a reset procedure. Since this sets the whole project back to the initial state, and not just one turtle, define the procedure in the project Procedures Tab.

```
to reset
  everyone [setspeed 0.1 setpos [0 0]]
end
```

Note that when you run the reset procedure, the turtles do not stay at position [0 0]. That's because they all arrive at the same point and then collide with each other. They react to the collision at the [0 0] position by moving away. You may want to create a reset button as well as part of your user interface.



Debugging - Escaping atoms

Sometimes an atom escapes the jar. This happens because **speed**, the input to **forward**, is larger than the thickness of the walls of the jar. If this happens, you could make the walls of the jar thicker or modify the bounce procedure to limit the speed increase:

```
to bounce
  if speed < 1 [setspeed speed + .1]
  bk 20 rt random 360
end
```

This means that if speed is less than 1, the speed will be increased a little. This way the atom won't "jump" over the wall.



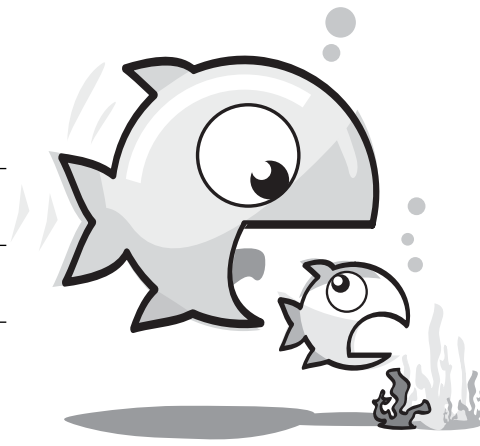


Challenges:

- Can you create a way of displaying the “temperature” of the atoms in the jar?
- What happens if the jar is smaller, larger or has a smaller/larger opening? Does this change the behavior of the atoms?
- Freeze the background of the page in order to preserve the jar with **freezebg**. Then set the turtles’ pen colors to any unused colors like yellow and put their pens down and observe the paths of the atoms. After a certain amount of time, clean up the trails by using the command **clean**, in the Command Center. You may want to add a clean button to make the process easier.
- What if one atom is bigger than others?
- Feel free to adjust the initial speed of the atoms if you are impatient.
- When you experiment with several atoms, something strange starts to happen. The atoms stuck on the jar’s mouth are blocking the way and the other atoms are bouncing off them (collision detection) instead of stopping. Change the instruction for the color pink, so that when an atom hits pink, it is moved to the side of the jar, for example: `clickoff setpos [300 0]` (use your own numbers)
- Mimic a nuclear reaction by “spitting the atoms” when they hit at high speed. If the speed is greater than a certain number, clone the atom and reset its speed:
clone who setspeed 0.1
- Create different bounce procedures (such as `atombounce` and `wallbounce`) for collision detection and color detection. For example, splitting the atoms or speed increase could occur only on collision detection while hitting a wall would only change the heading.
- Import the Stopwatch turtle (in the Turtles folder inside the MicroWorlds EX folder) and calculate the duration of your experiment with 2 atoms, 3 atoms, and so on.

Exploration 2 -

Environmental Chaos

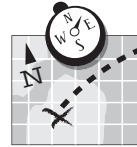


In this modeling project, you explore predator-prey relationships. Turtles play three roles in this simulation – predators, prey and food.

Feel free to enhance and improve on the project ideas suggested in this chapter. Be creative!

In this project you:

- Design a background
- Create the environment's inhabitants – food, prey, and predators
- Program the background
- Add interaction rules
- Run the simulation



Getting Started

Start by creating a new project. Name the project and remember to save often.

In this simulation, there exists:

- food. The food is eaten by
- small fish (the prey). The small fish are eaten by
- larger fish – the predators.

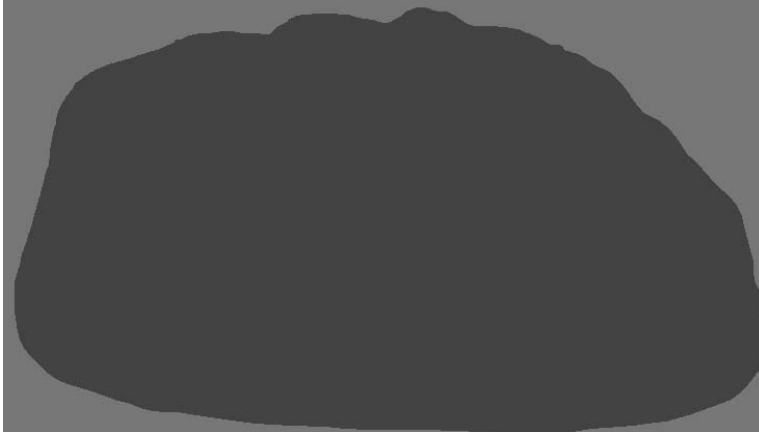
The predators also eat algae – they eat anything, so watch out!

Designing the background

The predators and prey in this simulation all live in and on the banks of a lake. So, first, set up the environment.

- Fill the screen with brown. Use the Painting tools to draw a large blue lake.
- When you are happy with your lake, type the following in the Command Center:
freezebg

This freezes the background image so you can restore your original image again, in case you draw something on it that you later want to erase.



- Save your project!

Defining the species

Since some turtles will be food, some prey (small fish), and some predators, it would be a good idea to assign a species property for all turtles to know who is who.

In the Command Center, type:

```
turtlesown "species"
```

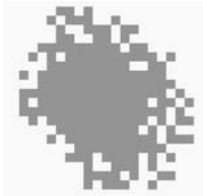
Create the food source

Think of the food as algae. Algae growth is represented by hidden turtles who move through the lake and at certain intervals stamp their shape on the screen, leaving food for the other creatures to eat.

Begin by creating one turtle in the role of food. Only after you have one turtle with all of the desired properties should you then copy/paste 3 or 4 duplicates.

Hatch a new turtle and open its backpack. First give it an algae-like shape:

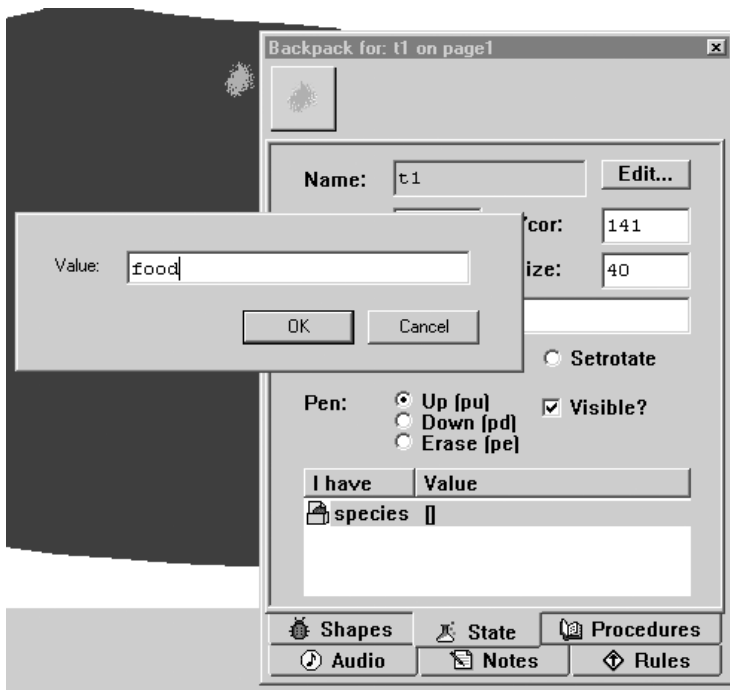
- Click on the Shapes Tab.
- Create a new algae-like shape. Use the spray can and dark green paint to make a food blob like the one illustrated below



- Click on the turtle to give it the shape.

Now, identify what it is:

- Click on the State Tab. Under the words I Have, double-click on **species**. Set the species to **food**



The food needs to “grow” at a specific rate. To have it “grow” on the lake, stamp the turtle’s shape. Then the turtle moves in some random way to a new location and stamps again.

- Click on the Procedures Tab
- Type the following procedure:

```
to grow  
stamp  
rt random 360  
fd random 20  
wait 100  
end
```

Next:

- Click on the Rules Tab
- Set the instruction for **OnClick** to:
grow
- Set it to Forever mode.

This means that food will be produced every 10 seconds.

Controlling the rate of growth - a **foodrate** slider

You can also control the rate at which the turtle stamps its shape by using a **foodrate** slider.

- Create a new slider
- Name the slider **foodrate**

- Set the minimum to 1 and the maximum to 20 and set the current value to a number between 1 and 20.

This slider represents the rate at which new food grows in the lake. These values were chosen arbitrarily and can be changed later.

Now edit the grow procedure to read:

```
to grow
stamp
rt random 360
fd random 20
wait 100 / foodrate
end
```



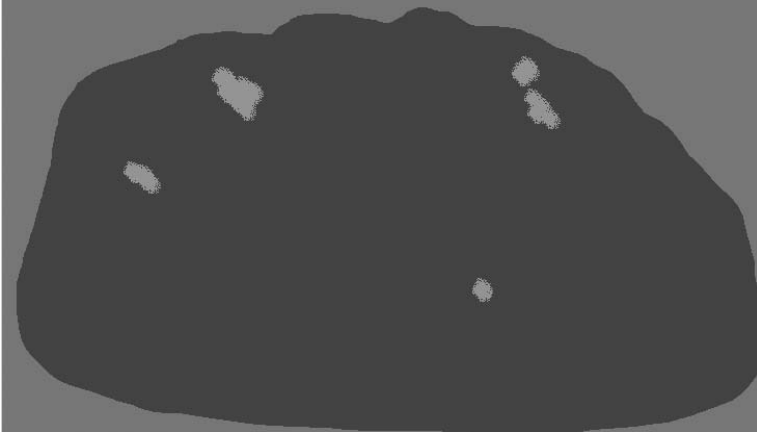
Important! Important! Important! Do not type a new procedure. Edit the procedure you already have.

If **foodrate** = 20, new food appears every 1/2 second. The food “grows” in a slightly random pattern.

Test your turtle by clicking on it. Does it multiply too quickly or not quickly enough? If so, adjust the **foodrate** slider.

Once you’re happy with the turtle:

- Select the food turtle
- Copy the turtle and paste one or two duplicates
- Drag the food turtles you create into different areas of the lake



Create the vegetarians

Now it's time to create the fish that eat the algae. Hatch a new turtle.

- In the Shapes Tab in this new turtle's backpack, create a friendly fish or fantasy shape for the turtle to wear, or drag clipart of a fish from the Painting/Clipart palette.
- Name that shape *fish*
- Open the State Tab of the turtle's backpack. Under "I have," double-click on the word **species**. Set the species to "prey. This will differentiate the fish who eat others and those who are eaten.

Moving

The prey-turtle does two things – it moves and it eats.

In order to control the turtle's speed, create a `prey_speed` slider. Set the minimum to 1 and decide on a reasonable maximum speed (10 sounds reasonable). You can always adjust this limit later.

- Give the turtle the following OnClick instruction in the Rules Tab of its backpack:

```
fd prey_speed wait 1
```

- Set the mode to Forever.

Test your fish. Adjust the `prey_speed` slider so that the fish is not swimming too quickly.

Eating

How does the prey-turtle eat the food? It stamps over it. Create a new shape for the prey-turtle. It will use this shape to “eat” food in the lake whenever the prey-turtle moves over the food. When the prey-turtle detects the color of the food, it will briefly change shape and stamp over the food with the color of water. Voila! The food disappears. You can even program the prey-turtle to get bigger as it eats.

- Create a second shape in the prey-turtle’s Shapes Tab. This shape should be a round figure in the same color as the lake water. Name that shape, eatstamp
- Type the following procedure in the Procedures Tab of this turtle’s backpack:

```
to eatfood
setsh "eatstamp
stamp
setsh "fish
  if size > 80 [burp]
  if size < 20 [hungry]
  if size < 100 [setsize size + 1]
end
```

- In the Rules Tab, set the **OnColor** instruction for dark green (or whatever color family you used for your food-turtle) to:

eatfood

Burp and **hungry** are recorded sounds. Record a sound for each, but keep in mind that these sound effects should be very short. Burp should remind the user how gluttonous the fish is. Hungry should indicate that the fish is still hungry. Make the icons for both sounds invisible.

Test your prey-turtle and your food-turtles. Are they growing, moving, and eating as you planned?

Respecting boundaries

By this time, you probably noticed that the fish are swimming all over the place – even out of the lake! To control this, program the color brown so that the prey-turtle bounces when it gets on land..

- Write the following procedures in the project Procedures Tab, since it will apply to all the fish, both prey and predators:

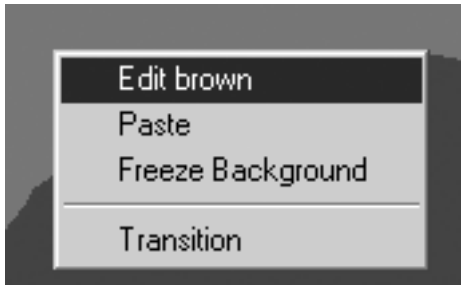
```
to bounce
bk 20
rt -30 + random 61
if not species = "food [shrink]
end

to shrink
if size > 5 [setsize size - 1]
end
```

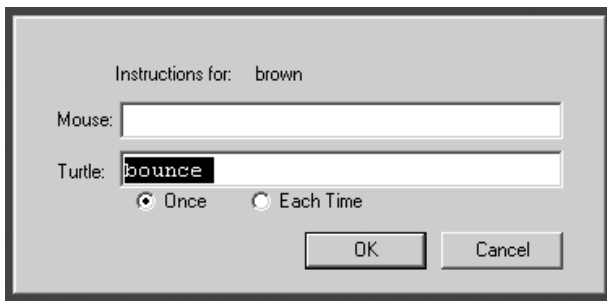
The **bounce** procedure causes any non-food turtle (the value for species is not food) hitting the brown exterior of the lake to back up and then turn either left or right by up to 30 degrees. Do you understand what the `-30 + random 61` is doing in that instruction? (Note that `rt -30` is the same as `lt 30`.)

If the turtle hitting brown is not food, it is reduced in size by 1 each time it bumps into the wall. This helps make the predator or prey weaker over time.

Right-click/Ctrl-click on the brown area on the screen.
Select **Edit brown** from the menu.



A dialog box opens. In the turtle instruction field, type:
bounce



Leave it set to Once.

Even if you're satisfied with how your prey-turtle responds in your simulation, do not duplicate it yet. Wait until you've completed all the programming for the predator-turtle.

Create the meat eaters

Now it's time to create predator turtles that consume the prey-turtles and algae, too.

Hatch a new turtle. Open the turtle's backpack.

- In the predator-turtle's Shapes Tab, create one or more shapes depicting a menacing creature. Also, copy the eatstamp shape from the prey-turtle and paste it in the Shapes Tab of the predator-turtle (or create a new eatstamp shape.)
- In the State Tab, under "I have", double-click on **species** and set the species property value to "predator"
- In the Rules Tab, set the predator-turtle's OnClick instruction to:
fd predator_speed wait 1

Set the mode to Forever.

- Create a new slider, **predator_speed**, with a minimum of 1 and a reasonable maximum speed.

Keeping track of bites

The predator behaves very much like the prey accept it can eat the prey. When the predator touches another turtle, it bites it. The predator grows a little bit when it eats algae and a lot when it bites a "fish" (a prey-turtle). In this simulation, if a prey-turtle is bitten more than four times, it dies.

In order to make sure the bites have the proper effect on the prey-turtle, the prey-turtle needs a way to keep track of how many times it has been bitten. A good way to do this is to create a special property only for the prey-turtle. Here are two ways to do this:

- 1) Click on the prey-turtle to make it the current turtle (if you started the prey-turtle's animation by mistake, you can stop it by clicking on the Stop All button). In the Command Center, type:

```
giveturtle "bites  
setbites 0
```



Look at the State Tab of the prey-turtle; you see the variable and its value.

- 2) Another way to make the prey-turtle the current turtle is to address it by its name. For example, if its name is t2, you can type:

```
t2, giveturtle "bites  
setbites 0
```



(Remember to use the name of your prey-turtle, if it is different. Also, you must use the comma after the name.)

Giveturtle is just like **turtlesown**, except it assigns the property to the current turtle only.



Eating – Part 2, Algae

Open the predator-turtle's backpack and click on the Procedures Tab. The following procedure is similar to the eatfood procedure in the prey-turtle's backpack. Type:

```
to eatfood  
setsh "eatstamp  
stamp  
setsh "dragon (use the name you gave to your menacing shape)  
if size > 80 [burp]  
if size < 20 [hungry]  
if size < 100 [setsize size + 1]  
end
```

In the predator-turtle's Rules Tab:

- Set the OnColor instruction to:
eatfood

Eating – Part 3, Biting Prey

When the predator touches another turtle, there are several things it must do. It first must determine the species of the turtle being touched. If it's prey, then the predator-turtle bites it. Other turtles (for example, other predator-turtles) are ignored. It then bites the turtle, triggering an action, a question and consequences, depending on the results of the question.

The action – the prey-turtle's bites increase by 1.

The question – how many total bites does the prey-turtle have?

The consequence – if the prey-turtle has 4 bites, it dies.

In order to carry out these three actions, the predator-turtle runs the bite procedure (Note: Some of the lines in the following procedures are long. Enlarging the backpack may help if you want to format the long lines).

```
to bite :victim
ask :victim [setbites bites + 1]
if (ask :victim [bites]) = 4
  [announce [You're toast - and yummy, too]
  ask :victim [setbites 0 clickoff ht]]
end
```



Ask temporarily “talks to” a turtle other than the current turtle, telling it to run a list of instructions.

Ask is used to ask the turtle specified (ask's first input) to run whatever is in the second input. If it's a command, ask acts like a command. For example:

```
ask :victim [setbites bites + 1]
```

Setbites is a command, so **ask** acts as a command.

If the second input is a reporter, **ask** acts as a reporter.

```
ask :victim [bites]
```

Bites is a reporter, so **ask** acts as a reporter.

The second input must be in square brackets.

Bite is a procedure that takes an input. Just like primitives can take an input (for example, forward), procedures that you define can also have inputs. The name of the input is written on the title line when defining bite. You can choose any name for the input (it could be victim, you, x, or any other word), but it must always be preceded by a colon. You can then use the input name (preceded by the colon) wherever you would want to use the value of that input when defining the procedure.



The title line of **bite** tells MicroWorlds that the procedure **bite** has a single input named **victim**. Then, in the body of the procedure, **:victim** is used as input to one or more commands, in this case, the primitive **ask**. When the procedure runs, whatever turtle name is used as input to **bite** will also be used as input to **ask**.

Eating – Part 4, Eating Prey

Finally, in order to eat, the predator-turtle runs the `bitefish` procedure. The `bite` procedure is a sub-procedure in `bitefish`. The predator-turtle checks what kind of turtle it is touching and, if it is a prey-turtle, it runs the `bite` procedure.

The predator-turtle also gets bigger.

```
to bitefish
  if (ask touchedturtle [species]) = "prey
    [bite touchedturtle
      if size < 40
        [setsize size + 2]]
end
```

As you can see, the input to `bite` is the reporter `touchedturtle`, which reports the name of the turtle being touched. The name of the turtle reported by `touchedturtle` is used everywhere `:victim` appears in the `bite` procedure.



The parentheses help MicroWorlds EX interpret the if statement correctly. Everything inside the parentheses is evaluated before being transmitted to the “equal sign”.

In the predator-turtle’s Rules Tab:

- Set the **OnTouch** instruction to:

bitefish

Test all your components. If you are satisfied with them, duplicate the predator-turtle and the prey-turtle several times.

Add a reset button

As always, having a reset procedure and button comes in very handy if you want to restart your simulation.

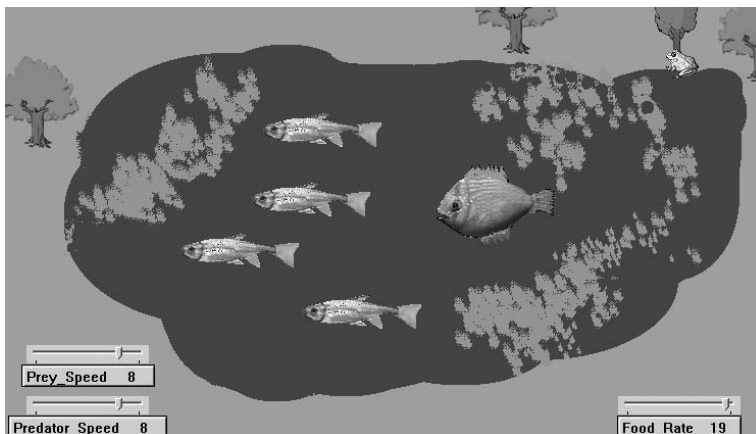
Since this is designed to reset the whole project, type the following procedure in the project Procedures Tab:

```
to reset
  setprey_speed 1
  setpredator_speed 1
  everyone [setsize 40]
  everyone [st clickon]
end
```

- Make a button with the instruction **reset**. Leave it in Once mode.

Run your simulation

- Click the **reset** button to set up the simulation
- Adjust the speed of the food, predator and prey to see what happens.

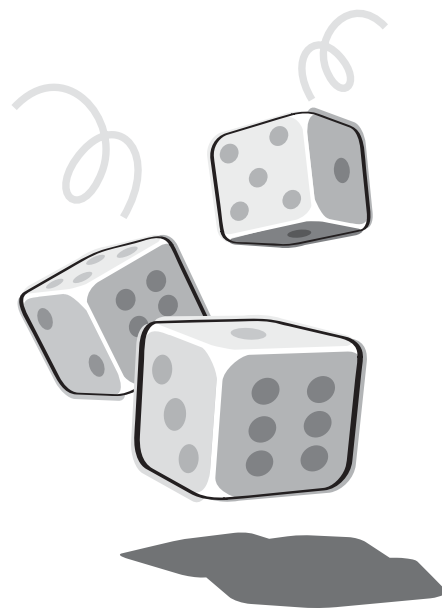




Challenges:

- Can you program the predator-turtles so that they are animated in more than one direction? Create shapes facing in the opposite direction and use `setrotate`.
- Can you program the predator's **bitefish** procedure so that it actually changes the shape of the prey when the prey is bitten? Perhaps a chunk could be taken out of the prey.
- Can you make the predator eat less, the fatter it gets?
- How might you add toxic waste to the lake?

Exploration 3 - Of Dice and Math



The theory of probabilities is nothing more than good sense confirmed by calculation – Laplace, 1796.

If you roll one die, each number (1, 2, 3, 4, 5, and 6) has an equal chance to be rolled, but if you roll two dice at once, each possible total for a roll (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) does not have the same chance of appearing. You have an equal chance of throwing a 2 as you do of throwing a 12 (1 chance out of 36), but what is the probability that you'd throw a 9 or a 7? What happens when you roll three dice?

You can use MicroWorlds EX to test your hypotheses. One of the benefits of using a computer to help you is that you can repeat a simulated throw of the dice a huge number of times quickly and without the computer getting tired.

In this project you:

- Design a die
- Roll the dice
- Create a record of all the throws
- Graph your results



Getting started

Create a new project.

Start by creating one die. When it is complete, clone it.

The die is really a turtle that is set to different shapes to show different sides of a die. Right-click/Ctrl-click on the turtle and select Open Backpack.

- Click on the Shapes Tab.
- In the first space, draw a frame shape. Then draw a dot or circle in the center of the shape to make the 1 side of the die.
- Click OK to save the shape
- Copy the shape and paste it into the second space on the Shapes Tab. Repeat this process so that the same shape appears in shapes 1-6
- Modify each of the six shapes to give each of them the correct number of dots.

Be sure that the die shapes are in the correct numerical order (the 1-side is shape #1, the 2-side is shape #2, etc.)



Randomly changing shape

Randomness is an important mathematical concept and is used in nearly every kind of game.



Try this:

```
setsh random 2
```

Did the turtle change shape? Try this several times.

You'll notice that sometimes the turtle is shaped like one of the die shapes and sometimes like a turtle. You may also notice that you never get a 2. That's because **random 2** reports a number (from 0 to one less than its input, 2) to **setsh**. So, it reports either 0 or 1. **Setsh 0** sets the turtle to the turtle shape.



In order to not get the turtle shape, you need a way of getting random to report 1 through 6. Try typing this in the Command Center:

```
setsh 1 + random 6
```

This instruction reports a randomly selected number from 1 to 6. It adds one to the number random chooses, which can be a number from 0 through 5.

Move the cursor back to this line and press **Enter/Return** again to see if the number changes. Try:

```
repeat 10 [setsh 1 + random 6 wait 2]
```

Does the die look more like a real die rolling now? You may want to change the input to wait to adjust the timing.

Rolling the die

Create a roll procedure to simulate a die roll. Open the turtle's backpack, if it is not already open.

Click on the turtle's Procedures Tab and type:

```
to roll  
repeat 1 + random 20 [setsh 1 + random 6]  
end
```

As you can see, **random** can even be used to determine how many times the shape should change when the die is rolled. This adds to the illusion that the die is rolling.

Type:

```
roll  
show shape  
3 Your number is probably different.
```

Try it a few times.

Once you're satisfied with your die, create a second and third die. You can do this using the Copy/Paste keys or menu items or you can use the command:

```
clone "t1
```

You should now have three dice. In the Command Center, type:

```
everyone [roll]
```

Did all the dice change?

Displaying Your Information

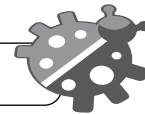
In MicroWorlds EX, there are many ways to display and use information. This project uses two possible presentation methods. There are also many ways to get MicroWorlds EX to automatically keep track of all the totals. One method is used in the project; a second is shown in the Extreme Challenges section at the end of the project.

Text box records

Create a text box for each possible total using one, two or three dice. Name each text box accordingly. For example, the first text box shows how many times 1 is rolled, so name the text box Got1 (You cannot use a number alone as the name of the text box, although names can have numbers in it.) Create 18 boxes and your setup will work with one, two or three dice.

Right-click/Ctrl-click on the text box and select **Edit**. Set the text box to single line (turning on the Grid helps). The box does not have to be large. It only needs to be large enough to show a 4-digit number. Once satisfied with your first box, copy it, once for each possible total, and edit each box. Then, type the number 0 in each box.

Don't write the word zero, use the number.



When you are finished you should have the following:

0	0	0	10	32	48
Got1	Got2	Got3	Got4	Got5	Got6
66	102	134	170	144	150
Got7	Got8	Got9	Got10	Got11	Got12
120	90	62	42	16	2
Got13	Got14	Got15	Got16	Got17	Got18

Text boxes act like variables. They each have a name connected to a value (the contents) that can change.



Anytime you create a text box, two dynamic primitives are also created. They're available in the project as long as the text box exists. The first is the name of the text box; it reports the text box's contents. The second is the word `set-` and the name of the text box; it sets the "value" (the contents) of the text box. For example, if you entered a 0 in the text box named `three`, type:

```
show got3
0
setgot3 1 + got3
show got3
1
```



If you did not enter a 0, you get an error message. The `+` sign needs to have two values to add together.

Rolling and Storing Your Results

Try this:

```
everyone [roll]
```

Each of the dice should change. Now type:

```
show t1's [shape]    Use the name of your turtle.
5                    Your number will be different.
```



Using an apostrophe with the name of a turtle gets the turtle to do whatever is in the square brackets that follow. (It is exactly like the `ask` primitive, which you may have used already.) If the square brackets contain a reporter, such as `shape`, this information is reported. In this case, it's reported to `show`.

In order to get the sum of the dice, you need to add together the numbers rolled – which are also the shape numbers – for the three dice. If each turtle reports its shape, you can then add them together.

Write a procedure in the project Procedures Tab to roll all three dice and calculate the sum.

Remember to use the parentheses. They help MicroWorlds EX interpret the statement correctly. Everything inside the parentheses is evaluated before being transmitted to the “+” sign.



```
to roll3
  everyone [roll]
  make "sum (t1's [shape]) + (t2's [shape]) +
    (t3's [shape])
end
```

Make creates a variable. **Make's** first input (sum), is the name of the variable. The second input is the value of the variable. In this case, the value equals the sum of all the turtles' shape numbers.



(A variable is like a box. The value of the variable is like the contents of the box. The same box can sometimes hold books and other times hold shoes. On the outside, though, it's still the same box. In **roll3**, sometimes the box – :sum – holds the number 8, sometimes 12, for example.)

Try the procedure:

```
roll3
```

Next, type;

```
show :sum
```

9 You probably will get a different number

The “dots” (:) before the word sum means that you don't want the word sum printed, but you want the value of the variable named sum (its contents – whatever it's representing).



Try it again.

```
roll3
```

```
show :sum
```

9 The value of the variable (:sum) changes with each roll.

Keeping Track

Finally, you need to keep track of how many times each sum is rolled. Each time a number is rolled, you need to add one (+ 1) to the corresponding text box.

In the project Procedure Tab, write a procedure to roll and store a number in the variable, and then add one to the correct corresponding box. (Note: You will never get a 1 or a 2 with three dice, but including them in the procedure means you can use this same procedure when rolling one or two dice, too.)

```
to keeptrack
  if :sum = 1 [setgot1 got1 + 1]
  if :sum = 2 [setgot2 got2 + 1]
  if :sum = 3 [setgot3 got3 + 1]
  if :sum = 4 [setgot4 got4 + 1]
  if :sum = 5 [setgot5 got5 + 1]
  if :sum = 6 [setgot6 got6 + 1]
  if :sum = 7 [setgot7 got7 + 1]
  if :sum = 8 [setgot8 got8 + 1]
  if :sum = 9 [setgot9 got9 + 1]
  if :sum = 10 [setgot10 got10 + 1]
  if :sum = 11 [setgot11 got11 + 1]
  if :sum = 12 [setgot12 got12 + 1]
  if :sum = 13 [setgot13 got13 + 1]
  if :sum = 14 [setgot14 got14 + 1]
  if :sum = 15 [setgot15 got15 + 1]
  if :sum = 16 [setgot16 got16 + 1]
  if :sum = 17 [setgot17 got17 + 1]
  if :sum = 18 [setgot18 got18 + 1]
end
```

Make sure all the number boxes are set to 0.

Now try the **roll3** and **keeptrack** procedures a few times.

Do you notice the values in the boxes changing?

Repeatedly Repeating

One task at which the computer is great is repeating an experiment thousands of times without getting tired. The computer can roll the dice as many times as you want.

Reset all the boxes to 0.

Type:

```
repeat 100 [roll3 keeptrack]
```

Do you get the results you expected?

It would be interesting to try repeating the keeptrack procedure different numbers of times to see if you get the same results each time.

Using a slider is one way to easily change the input to repeat. Create a slider and call the slider something like Times. Set the minimum to 1 and the maximum to 5000.



Now, reset all the text boxes, set the slider, and try your experiment:

```
repeat times [roll3 keeptrack]
```

Create a button with that instruction to run the experiment. The button's label should tell about what it does. For example: *Experiment 3 dice*

Set the mode to Once.

Reset all your boxes and try your experiment several times.

Resetting Everything

It would be easier to reset all the boxes with one command, rather than one-by-one. In the project Procedures Tab, write a procedure that resets all the text boxes to 0.

```
to reset
  setgot1 0
  setgot2 0
  setgot3 0
  setgot4 0
```

.

.

You fill in this part

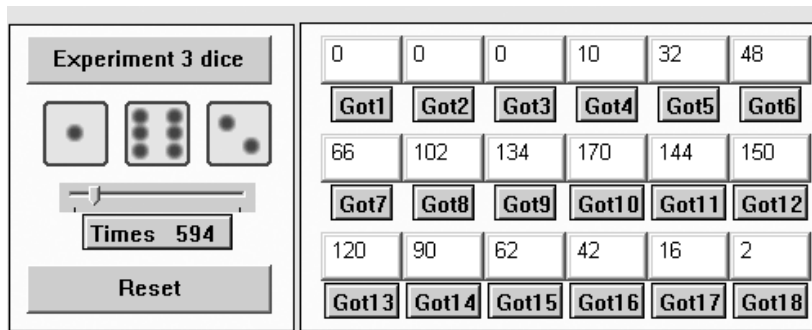
.

.

```
end
```

Make sure you reset all the boxes. Create a Reset button to run this procedure.

Your project may look like this:



Testing Your Hypothesis

Run your experiment several times. Are the results the same as what you predicted? Do the results change the more you roll the dice?

Graphing Your Results

Although all your results are visible on the screen as numbers, you may be better able to understand the results if you graph them. Create a bar graph to show which sums are rolled most frequently.

Add a new page to your project. Draw an x and y axis on the screen (turning the Grid on will help you with the page layout).



Freeze your background:

freezebg

Use a turtle to represent each possible sum. The turtle will draw a line based on the number of times the sum is rolled.

Start by creating one turtle and placing it on the x-axis.
Place it 20 steps from the y-axis.



Give the turtle a new property - a homebase.

```
giveturtle "homebase
```

Open the turtle's backpack. In the turtle's State Tab:

- Put the turtle's pen down
- Then check the turtle's current position (xcor and ycor).
- Double-click on the word **homebase** under the heading "I have".
Set **homebase** to a list containing the turtle's current xcor and ycor positions.

In the turtle's Procedures Tab, type the following procedure:

```
to drawline  
fd 2 * got1  
setpos homebase  
end
```

The turtle puts its pen down and goes forward two times whatever value is in the got1 text box on the other page. It then returns to its homebase.

Try your turtle. Type a number in the got1 text box and type this in the Command Center:

```
setpos homebase  
drawline
```

Did your turtle draw a line in the correct place? To erase this line, use:

```
clean
```

If you're happy with your turtle, copy and paste a new turtle. Edit the turtle in the following ways:

1. Change the **homebase**. Double-click on the value of **homebase** in the new turtle's State Tab and change the xcor so that the next turtle is 20 steps to the right.
2. Change the **drawline** procedure so that it uses the value reported by the text box named got2:

```
to drawline  
fd 2 * got2  
setpos homebase  
end
```

Repeat this process, creating one turtle for each of the possible sums. You should have 18 evenly spaced turtles, each with a slightly different drawline procedure.



Test the turtles:

```
everyone [setpensize 10]
everyone [drawline]
```

Each turtle uses its own **drawline** procedure – the one that’s in its backpack.

Add two buttons, one to draw the graph (using the above instruction) and one to clean the graph. You may also want to label the x and y axis, and use the Painting tools to give a different colors to all the turtles.

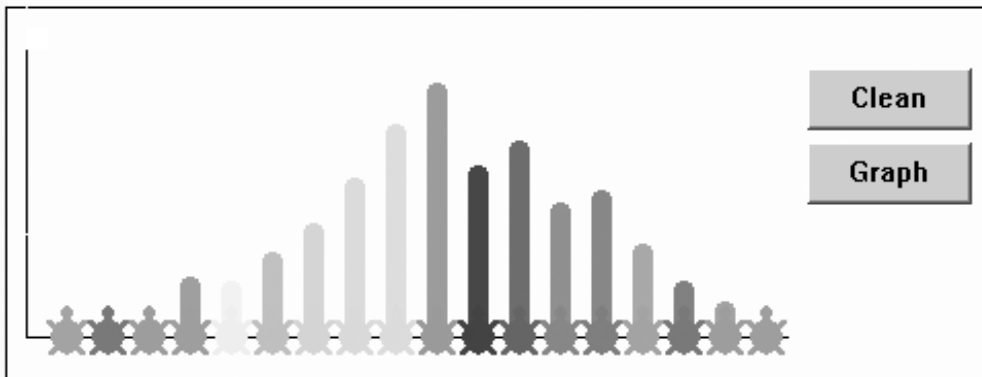
Your screen may look like this:



You can also hide the turtles:

```
everyone [ht]
```

Run your experiment and then click the Graph button to record your results.



Challenges:



- Run the experiment with 1 or 2 dice. All you need is an extra button that rolls the right number of dice. You also need to create a roll1 or roll2 procedure. (You have too many text boxes for this experiment, but it doesn't matter.)
- The numbers you get on page1 of your experiment are called raw scores. If your times slider is set too low, the graph will be too flat. If it is set too high, the bars may exceed the length of the y axis. Convert raw scores to percentages to get values between 0 and 100 no matter the number of times you throw the dice.

Write a procedure in the project Procedures Tab to convert the scores to percentages.

It could look like this:

to convert

```
setgot1 int (100 * got1 / times)
```

```
setgot2 int (100 * got2 / times)
```

```
setgot3 int (100 * got3 / times)
```

```
setgot4 int (100 * got4 / times)
```

•

• You fill in this part.

•

end

Add a Convert to Percentages button on page1.

When graphing the results as percentages, you will probably find that the bars on the graph are too short. Do you know how to change this?

- Change your dice. Try giving one die two sides with the same number of dots. You will need to adapt your roll procedure, since the shape number will not be the same as the number rolled. Here's one way you can do it. In this example, shape two - the side with 2 dots - is changed to a shape with 4 dots. Your procedure could be:

```
to roll
repeat 1 + random 20 [setsh 1 + random 6]
if shape = 2 [setsh 4]
end
```

Do you understand why this works? If 2 is rolled, the shape of this turtle is reported as 4.

How does this change the outcome of your experiment?

- Add a fourth die. Change your set up and procedures accordingly.



Extreme Challenges

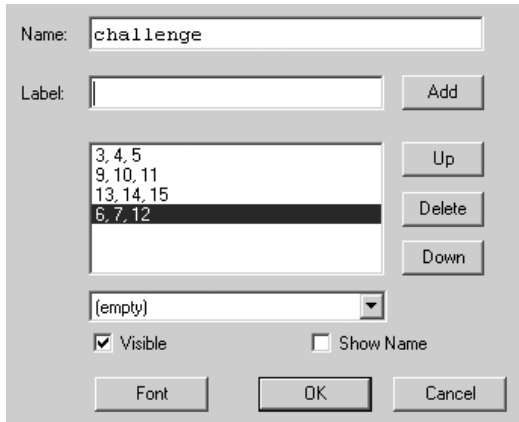
- Before your friends try your experiment, ask them "Which numbers will be rolled the most before trying the experiment?" After the experiment is run, your friends can compare their guess to the actual results.

One way to ask this question is to create a dialog box that contains the question plus multiple answer choices.

In the Dialogs menu, select New Dialog Box. The Dialog Editor opens.

You can add text boxes and choosers in your dialog box.

- 1- Add a text box in which you can write your question. If you want, make the text box transparent.
- 2- Next, add a Round button set. In the round button set dialog, give the Round button set a meaningful name, such as challenge. In the field next to Label, type one of the answer choices. Click Add. Repeat these last two steps for each of the choices. It may look like:



Keep the default value set to (empty). This means none of the choices will be selected when the dialog box is first displayed.

- 3- Arrange the text box and the Round button set close to the OK and Cancel buttons in the bottom right hand corner of the dialog box. These buttons are automatically added to your text box. Drag the little black squares on the edge of the editor screen to reduce the size of the dialog box so that there isn't too much blank space.
- 4- Click OK (the one at the top of the Editor) when you're finished.

To test your dialog box, type:

dialog1 (use the name of your dialog box)

A dialog box should open and your question should appear.
Select an answer and click OK.



Type:

```
show getlabel "challenge
```

```
6, 7, 12 (this depends on which answer choice you selected).
```

Getlabel gets the contents of the label of the selected choice in challenge.

5- Add a text box on the graphing page that displays the guess (for example, the text box could be named guess).

6- Add a button to your first screen to display the Dialog box.

The instruction could read:

```
dialog1 setguess getlabel "challenge
```

Getlabel "challenge reports its information to **setguess**, which sets the contents of the guess text box.

- Try writing a very short procedure to record all the rolls in text boxes.

A one-line procedure could look like:

```
to keeptrack  
run list (word "set "got :sum) 1 + run  
(word "got :sum)  
end
```



Run executes whatever instruction is in the word or list that is its input.

In this case, you must use a list as input, since the instruction has several components. The first component is the word created by set, got, and the number rolled (the value of sum) – for example, if you roll 12, the word would be setgot12. The next component is the outcome of 1 + whatever is reported by the word created with the instruction **word "got :sum**. You must use **run** before **word "got :sum** so that a number is reported, not the word.

See if you can write a short procedure to graph the information using only one turtle.

Section 2

Videogame Design

Videogames are great to play, but even more fun to build. With MicroWorlds EX its easy to design your own games and share them with friends. These games can even be played on the World Wide Web via the free MicroWorlds EX Web Player.

You will build two simple projects based on popular videogames, Frogger™ and Pacman. These are excellent games to build in MicroWorlds EX because of their simplicity and room for improvement. You should be able to enhance the games and customize them to suit your own goals.

These two videogame projects offer an opportunity to introduce a number of more sophisticated (but not too difficult) programming concepts.

What's in a game?

Videogames all include the following elements. You will explore each of them during the construction of the projects.

- **Logic**

What are the rules of the game? How does a successful player think while playing the game? What sorts of strategies are most effective? Does the player play against herself? The computer? Another human player? If the computer plays a human contestant, how will you teach the computer to play the game?

- **User Interface**

What buttons does the user have to click? Which keys do they have to press? Are there instructions available for new players? Does the screen layout make sense to the user?

- **Motion**

Videogames differ primarily from other types of game in that things move in videogames. These games are often tests of eye-hand coordination more than they are of logic and strategy.

- **Collision detection**

Crash! Boom! Pow! Videogames are fun because things crash into other things: cars into walls; vehicles into other vehicles; projectiles into people, buildings or vehicles. You earn points for hitting things and lose the game when certain things touch you. The MicroWorlds EX environment can be programmed to interact with your moving objects and for the moving objects to interact with each other.

- **Scoring** (data-keeping)

There are all sorts of ways to keep track of important data and success levels – game level, points, collected treasure, etc... Your games will probably include some form of scoring.

Motion and color detection

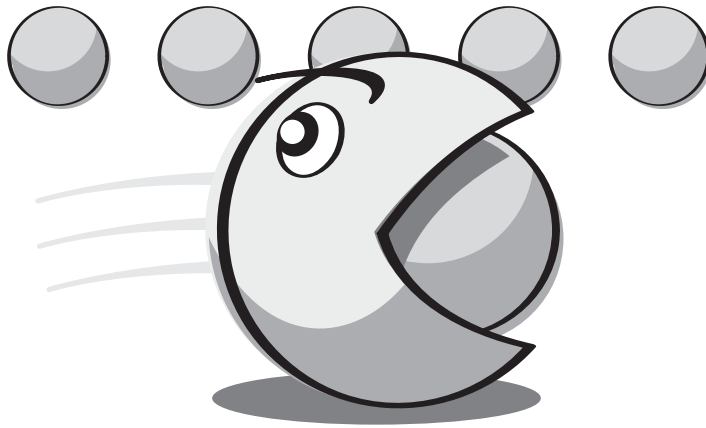
As you saw in the previous projects, MicroWorlds EX can use colors painted on the screen to detect when a turtle is over that color.

That should illustrate how easy it is to have a piece of the graphical background influence the behavior of a turtle in MicroWorlds EX.

You can also program the color to react when the mouse is clicked on it by typing instructions in the mouse instruction field of the color's edit box.

Game 1 -

Snacman



Our simple Snacman game has three kinds of players – Snacman, ghosts and food pellets. Each of these players will be represented by turtles. All of the game play takes place within a maze, so you also need to create the maze that the characters inhabit.

The goal of Snacman is to munch around a maze, eating all of the food pellets before a ghost makes you disappear. You have to try and outsmart and out-manuever the ghosts.

In this project you:

- Draw a maze
- Design Snacman, spooky ghosts, and food pellets
- Create navigational controls
- Program interactions with the background (maze)
- Program interactions between Snacman and ghosts and food
- Track and record the score
- Add final touches to the user interface
- Play the game



Getting Started

Create a new project. Use the Painting tools to draw a dark blue maze taking up much of the page.

Be sure:

- to use dark blue as the paint color and make the walls about the width of a turtle shape.
- that there is enough “white space” between the walls so the turtles can move through the maze without touching the walls.



Design Snacman

Hatch a new turtle and turn it so it is facing east (right). Open its backpack.

- In the Shapes Tab, design two turtle shapes – *rightopen* and *rightclosed*



- In the State Tab, name the turtle Snacman and type *rightopen* *rightclosed* into the Shape field, or select both shapes in the Shapes Tab and then click on the turtle

- In the Rules Tab, in the **OnClick** instruction field, type:

```
fd 2 wait 1
```

- Choose the Forever mode.

Click on the turtle. Does Snacman move and munch? If not, check that you followed the instructions above correctly.

Program the maze walls

There needs to be a way to keep the turtles within the closed maze. Program the dark blue so that a colliding turtle backs up and turns right or left at random.

- Right-click/Ctrl-click anywhere on the dark blue walls of the maze
- In the instruction field next to Turtle, type **bounce**
- Leave the mode to Once and click OK

Now, create a bounce procedure in the project's Procedures Tab.

Type the following procedure:

```
to bounce
bk 20
ifelse 1 = random 2 [lt 90] [rt 90]
end
```

This procedure causes the turtle to back up and then flip a coin to randomly decide if it should turn left or right, adding a bit of spontaneity to the game.

Ifelse works in a similar way to **if**, except it has both a true and a false consequence as the result of the condition being checked – in this case whether a 1 or 0 was picked at random. If a 1 is picked, **ifelse** runs the instructions in the first list; if 1 is not picked, it runs the instructions in the second list.



Check it out!

Put your Snacman turtle in the maze (center of an aisle is best). Click on it and watch what happens when it hits a wall

Does it back up and turn at random?

Ghost!

The ghost just wanders around the maze until it collides with Snacman.

Hatch a new turtle and open its backpack:

- In the State Tab name the turtle “Ghost
- In the turtle’s Shapes Tab, create two ghost shapes (you should know how to do this by now)
- Put both shapes on the turtle or enter their shape names in the Shape field of the turtle’s State Tab
- In the Rules Tab, give the turtle the following OnClick instruction:
fd 1 wait 1
- Choose Forever

The ghost initially moves slower and in a dumber fashion than Snacman so the game is easier for the human player. Put the ghost-turtle in the maze, click on it and see if it behaves properly.

Spookier movement

It would be cool to program the ghost to move in a more ghostly way than Snacman. You can do this by making it turn more randomly when it bounces into a wall. This also adds an air of unpredictability for the human player.

Right-click/Ctrl-click on the ghost turtle and choose **Open Backpack**

- Click on the Procedures Tab in the backpack
- Define a bounce procedure for the ghost

```
to bounce
bk 20
rt random 360
end
```

Now whenever the ghost hits dark blue on the page, it will back up a bit and turn randomly. If any other turtle hits the wall, it backs up 20 turtle steps and turns either right or left by 90 degrees. Each turtle bounces in its own way - runs its own version of the bounce procedure - when it hits the wall.

Flickering ghost

You can make the ghost flicker while it moves by adding a **setopacity** command to the **OnClick** instruction in the backpack.

Change the **OnClick** instruction to:

```
setopacity random 100 fd 1 wait 1
```

Setopacity sets how opaque (or transparent) a turtle is.
Setopacity 0 makes the turtle completely transparent.



Food pellets

The job of the food pellets is to just lie around waiting to be hit by another turtle. If that turtle is Snacman, the food pellet will be eaten. If it is any other turtle, nothing happens and the food pellet can go back to sleep.

Hatch a new turtle and open its backpack.

- In the State Tab, name the turtle “food.”
- In the turtle’s Shapes Tab, design a food pellet shape for it to wear. It doesn’t need to animate so one shape is sufficient.
- Put the shape on the turtle

Clone the turtle

You need lots of food pellets to be in the maze, so make copies of the one you just created. Use either one of the Copy/Paste methods or the clone command.

```
repeat 10 [clone “food]
```

All the pellets are stacked one over the other. Drag each of the pellets to the desired location.

You may change the input to **repeat** to clone as many pellets as you wish. Just make sure that a food pellet is not hiding behind another.

Add some navigational control

How do you steer Snacman as he moves? One way is to create four clickable turtles to use as your joystick.

Hatch a turtle and open its backpack.

- Click on the Shapes Tab and create an up arrow shape or get one from the Painting/Clipart palette.
- Click on the Procedures Tab in the turtle's backpack and type:
to move.up
broadcast 0
end
- Click on the Rules Tab and in the **OnClick** instruction field type
move.up
- Create three more arrow-turtles, one for each direction. Type one of the following procedures in the Procedures Tab of each turtle and set each turtle's **OnClick** instruction accordingly.

For the right arrow turtle, type:

```
to move.right  
broadcast seth 90  
end
```

For the left arrow turtle, type:

```
to move.left  
broadcast seth 270  
end
```

For the down arrow turtle, type:

```
to move.down  
broadcast seth 180  
end
```



Broadcast is a command that takes an input and yells that input to any turtle that may be listening. You need to program Snacman so that when the arrow-turtles yell out, Snacman gets the message and turns in the right direction.

- Arrange the arrow-turtles in the form of a joystick.

Telling Snacman where to go

You can “catch” a broadcasted message and act upon it by using the OnMessage field in the Rules tab of Snacman’s backpack.

- Open Snacman’s backpack and click on the Rules Tab
- In the OnMessage instruction field, type:

seth message



Now when a number is broadcast by an arrow-turtle, message reports the number to **seth** and Snacman automatically turns in the indicated direction.

Eat or be eaten

When Snacman hits another turtle, it either eats (if the other turtle is a food pellet) or gets zapped (if the other turtle is a ghost). If Snacman gets zapped, the game is over. Write two procedures, one for being zapped and one for eating.

This is a good time to add some sound effects. Record a short sound Snacman will make when it eats a pellet. Name the sound yum and hide the icon if you prefer. (If you don’t have a microphone connected to your computer, compose a one note melody with the music tool and name it yum.)

Open Snacman's backpack.

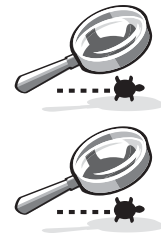
- Click on the Rules Tab.
- Type the following procedures into the Procedures Tab of Snacman's backpack:

```
to zapped
  everyone [clickoff]
  announce [Game over!]
end

to eat
  launch "yum
  ask touchedturtle [ht]
end
```

The primitive, **touchedturtle** reports the name of the turtle with which the current turtle has collided.

Ask lets you talk to another turtle briefly and then go back to talking to the current turtle.



Ask has two inputs. The first is the name of a turtle (or text box) – in this case, whatever turtle name is reported by **touchedturtle**. The second is a list of instructions, enclosed in square brackets. **Ask touchedturtle [ht]** asks the turtle hit by Snacman to hide – to disappear.

Next, create a procedure to check who the Snacman is touching:

```
to whoareyou
  ifelse touchedturtle = "ghost [zapped][eat]
end
```

The procedure says that if the turtle you're touching is the ghost, you'll be zapped. Otherwise, eat a pellet.

In the Rules Tab of Snacman's backpack, type the following in the **OnTouching** instruction field:

whoareyou

When Snacman touches another turtle, it checks who it is and then does the appropriate action.

Extension: Can you add sound-effects or an animation to the zapped procedure?

Keeping track of strength

Use text boxes to keep track of Snacman's strength during your game. Each pellet Snacman eats is worth 1,000 points (Be generous!). When Snacman eats a pellet, his strength increases. Can you beat your high strength?

- Create a new text box on the screen. Make sure it is not in the maze or over the navigation buttons.
- Right-click/Ctrl-click on the text box and select **Edit**. Name the text box, strength.
- Select the **single line** mode.
- Choose a font, size, color and style for the text box
- Initialize the strength by typing **setstrength 0** in the Command Center. This is a common practice when programming since you later will want to add a number to the value in the box. You can't add a number to emptiness.
- Open Snacman's backpack and click on the Procedures Tab.
- *Change* the eat procedure by adding a **setstrength** line.

```
to eat
  launch "yum
  ask touchedturtle [ht]
  setstrength strength + 1000
end
```

**Important! Important! Important! Do not write a new eat procedure!
Change the existing one:**



When there is a text box named **strength**, **setstrength** changes its contents.
Now, each time Snacman eats, he gets stronger – a lot stronger!

Keep track of your turtles

Arrange the turtle, ghost and food pellets so that they are where you want them to be at the start of each game. Then teach MicroWorlds EX to remember where they are so you can easily reset them with each new game. Be sure that the turtle and ghost are as far away from each other as possible.

Give all the turtles a new property – **homebase**:

- Type the following instructions in the Command Center:

```
turtlesown "homebase  
everyone [sethomebase pos]
```

The **everyone [sethomebase pos]** line tells each turtle to remember its current position and set it as the property called **homebase**. Look in the State Tab of Snacman's backpack, under the heading I Have, to see what properties the turtle owns.



- Since this is a procedure for the whole project, write the following procedure in the project Procedures Tab:

```
to setup  
everyone [setpos homebase st]  
end
```


Test the **setup** procedure by doing the following:

- Move some turtles around the screen
- Type setup in the command center

Did the turtles march back to their proper places?

Putting it all together

You are just about to play your first game of Snacman. There are just a few jobs left to do.

First, you need to know when the game has been won by the player. The player wins when he eats all of the food pellets before being zapped by the ghost.

- Count the number of food pellets in your maze
- Multiply that number times 1,000
- Remember that number
- Create a new button with the instruction, play
- Write a play procedure in the project Procedures Tab

```
to play  
setup  
setstrength 0  
everyone [clickon]  
when [strength = ____] [announce [You're a  
champion!] stopall]  
end
```

Put the number you get by multiplying your number of pellets * 1000 in the blank found in the when instruction.

When is a command that takes a conditional list as input and waits until that condition is true. When it is true, the list of instructions in the second set of brackets is run. **When** runs in the background as a process until you stopall or cancel the process. **cancel** [**setstrength** = ____] will do the trick. Be careful using **when**. It should be used sparingly as it may clutter up memory or confuse you while debugging. Click on the Processes Tab to see a when process in action. You may Right-click/Ctrl-click on it to pause or delete the process.



If you find that either the ghost or Snacman move too quickly or slowly, adjust the fd or wait commands in the Rules Tab of their backpacks.

No cheating! If you freeze Snacman and the ghost, they will be controllable using the arrow buttons and commands, but the player will not be able to drag them around the maze using the mouse. In fact, you should freeze the entire page so the arrows don't "move" by accident.

Click on the play button and have fun!

Challenges:



- Keep track of how long it takes for you to win the game.

MicroWorlds EX has a built-in timer. The timer counts in tenths of a second.

Resett sets the clock to 0



Timer reports the current time since the last reset, in tenths of a second.

Start the timer when the game begins, keep track of how many seconds pass and calculate a score based on how quickly you munch food pellets.

Create two text boxes, score and seconds. They should look like the strength text box you already created. Find a good place to put them on the page.

Edit the play procedure to initialize two new text boxes, score and seconds:

```
to play  
setup  
setstrength 0  
setscore 0  
setseconds 0  
everyone [clickon]  
when [strength = ____] [announce [You're a  
champion!] stopall]  
end
```

- 1- Open Snacman's backpack
- 2- Click on the Procedures Tab
- 3- Create the following procedure:

```
to track.time  
reset  
forever [wait 10 setseconds timer / 10  
setscore strength / seconds]  
end
```

This says to wait one second (ten tenths of a second), display the time (in seconds) in the seconds text box (timer divided by 10), and set the score in the score text box.

Your score will be based on the strength divided by how long it took to eat that many pellets!

- Make your Snacman munch in the proper direction

You can make Snacman munch in the direction he moves, if you create three more sets of open and closed mouth Snacmans in the Shapes Tab of the Snacman turtle's backpack.

- 1- Design six new turtle shapes with the following names. Be sure to name them! You can copy, paste and then rotate existing turtle costumes to make life easier.

upopen - upclosed

leftopen - leftclosed

downopen - downclosed

- 2- Write the following munch procedure in Snacman's Procedures Tab:

```
to munch
if heading = 0 [setsh [upopen upclosed]]
if heading = 90 [setsh [rightopen
rightclosed]]
if heading = 180 [setsh [downopen
downclosed]]
if heading = 270 [setsh [leftopen
leftclosed]]
fd 2 wait 1
end
```

- 3- Change the **OnClick** instruction for Snacman to:

munch

Now your Snacman should munch in all four directions. Feel free to accelerate or slow down Snacman and the ghost if you think it will make the game more exciting.



Extreme Challenges:

- Create a special “power pellet” that either zaps the ghost or makes Snacman immune to its destructive powers for 5 seconds.
- Can you make a “Level” slider that controls the speed of the game? Can you program the game to automatically reset the pellets and go to the next level when the current level is completed? You may want to create the second level on another page.

Game 2 –

Squish the Turtle:

A Web-ready Game



In “Squish the Turtle” the player needs to get a turtle across heavy traffic without being squished. Does this remind you of any video games from the past?

In this project you should make the page smaller so that the game can be played via the World Wide Web. Smaller projects take less time to load via the web and tend to work better online.

In this project you:

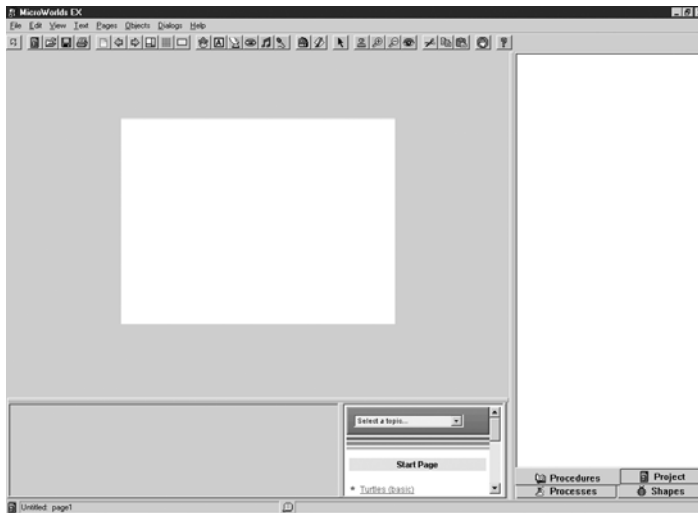
- Draw the background scene
- Create your turtle, Fred
- Create navigational controls
- Create traffic
- Add collision detection
- Program interactions with the background
- Add traffic signals
- Share your game with others



Getting Started

Create a new project.

- In the File menu, choose **New Project Size-MicroWorlds Small**



- Paint a background image like the following:



Add the Turtle

Hatch a turtle. Drag the turtle to a position centered in the green area slightly below the road. Open the turtle's backpack and, in the State Tab, name the turtle *Fred*

Create a slider

Next, create a slider to control Fred's speed.

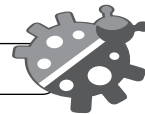
- Create a slider
- Name the slider speed with a minimum of 1 and a maximum of 40

In Fred's backpack:

- Click on the Rules Tab. Type the following **OnClick** instruction:
fd speed / 10

This causes Fred to not go too fast by dividing the speed by 5.

Remember to put a space on each side of the division sign.



Set it to Forever mode.

Navigating

Use buttons to navigate Fred across the street. Clicking on the buttons cause Fred to turn while still moving forward. Once Fred crosses the street, he returns back to the green field and the game gets faster. The player who is able to get across the street with the fastest speed is the champion.

- Write the following procedures in the project Procedures Tab:

```
to north  
fred, seth 0  
end
```

```
to east  
fred, seth 90  
end
```

```
to west  
fred, seth 270  
end
```

```
to south  
fred, seth 180  
end
```

- Create four buttons, each with one of the following labels and instructions:
north or **south** or **east** or **west**
- Choose Once and click OK
- Drag the buttons in the shape of joystick controls to the right-hand side of the playing field.

Create a traffic jam

Hatch a turtle.

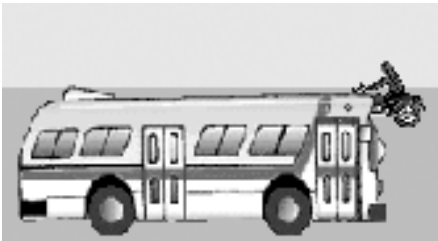
- Turn it either to the right or left.
- Create or drag vehicle shapes into the project Shapes Tab so the shapes may be shared by multiple turtles. Put a vehicle shape on the turtle.
- Open the vehicle-turtle's backpack. Click on the Rules Tab. In the **OnClick** instruction field, type:

fd speed / 10 or

fd speed / 8 to make slower and faster vehicles.

Clone the vehicle by selecting, copying and pasting duplicates all over the roadway or use the **clone** command. Be sure to turn the vehicles in the correct direction. Use the Shape editor to create flipped images if needed. If the vehicles are too big, use the magnifiers or the command **setsize** to make them smaller. Feel free to have different vehicles use different shapes (cars, trucks, motorcycles, etc...).

Move the vehicles around to see if the ones on this side of the street are in front (hiding others). If they aren't, right-click on the turtle and choose **In Front** or **In Back**. (You may have noticed that, in fact, new turtles are created in front of the older ones.)



What Happens When Fred “Meets” a Vehicle

If Fred is hit by a vehicle-turtle, Fred is squished.

Open Fred’s backpack

- Click the Procedures Tab and type:

```
to check
  announce pick
    [[I’m squished!]
    [Game Over, Man!]
    [Ooh, I hate when that happens!]
    [That really hurts!]
    [Ouch, ooooh, eeetch!]]
stopall
end
```



In this procedure, the **pick** command randomly picks an item from the big list that contains several shorter lists. The **announce** command then displays what is picked. The long instruction can be typed all on one line, but it is easier to read if you format it on several lines.

- Click Fred’s Rules Tab
- Type the following in the **OnTouching** field:
check

Setting up

Time to set up the game. First, put Fred at the position where you want him to start.

Give Fred a new property – **homebase**. To give Fred this variable, type in the Command Center:

```
fred, giveturtle "homebase
```

To give the variable a value, type:

```
sethomebase pos
```

Giveturtle is just like **turtlesown** except **giveturtle** assigns the new value or property only to the current turtle. **Sethomebase pos** sets Fred's **homebase** value to his current position.



The setup and play procedures below should be typed in the project's Procedures Tab since they are used by the project and not just one turtle.

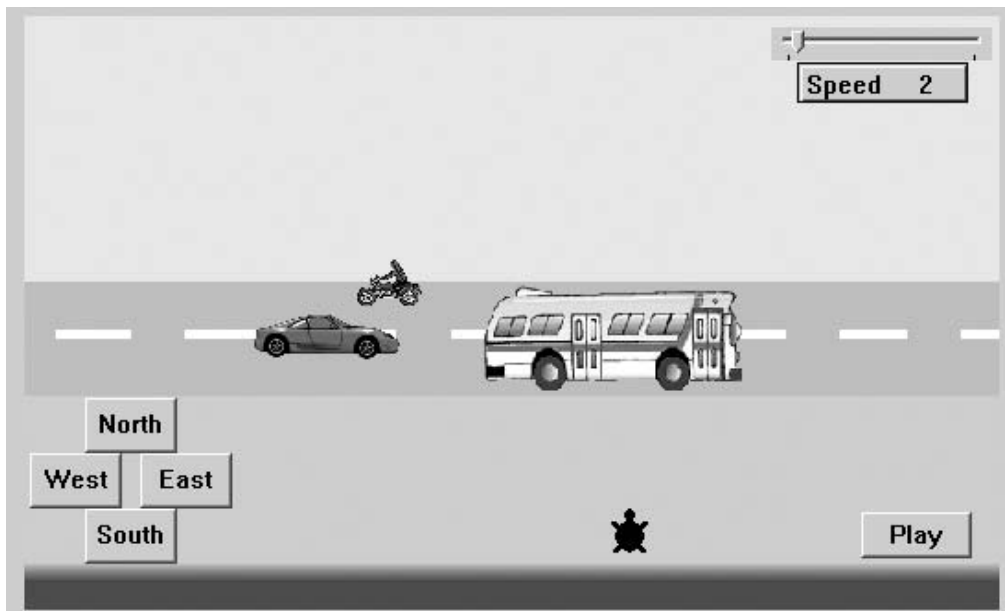
```
to setup  
fred, setpos homebase  
seth 0  
setspeed 1  
end
```

Since there is a slider named *speed*, **setspeed 1** sets the value of that slider to 1.

Play causes everything to reset before asking everyone to go.

```
to play  
setup  
everyone [clickon]  
end
```

Add a play button.



Next level

Now program Fred so that when he reaches the sky he goes back to his homebase and picks up the pace for a more challenging game.

Open Fred's backpack.

- Click on the Rules Tab
- Select the sky blue color (or whatever color you used for the sky) in the OnColor menu. Type the following **OnColor** instruction. Leave the mode set to Once.

```
setpos homebase setspeed speed + 1
```

In fact, the speed slider is used to control the speed of the objects, but it also indicates the "level" of the game, or your score if you will.

The traffic light

Adding a turtle that looks like a traffic light can add another level of control for a player.

- Hatch a new turtle, name it light and drag it to an open space (off the road).
- In the Shapes Tab of light's backpack, design two shapes, one of a traffic light with a green light illuminated and one of a traffic light with the red light illuminated.
- Name the shapes, *greenlight* and *redlight*, respectively
- Type the following procedure in light's Procedures Tab:

```
to switch
  ifelse shape = "greenlight
    [setsh "redlight broadcast "red]
    [setsh "greenlight broadcast "green]
end
```

- Set the **OnClick** instruction to switch
- Write the following procedure in the project Procedures Tab

```
to checklight
  if message = "red [clickoff]
  if message = "green [clickon]
end
```

- Open the backpack of each vehicle turtle. Click on the Rules Tab and, in the **OnMessage** instruction field, type:

```
checklight
```

The **checklight** procedure needs to check each possible message that can be broadcast. In this case, there are only two, red and green. You could use either **ifelse** or **if**. If there are exactly two messages being checked, it is better to use **ifelse**.

Now, the vehicles will start and stop as the traffic light is clicked.



Because the **play** procedure simulates a click on every turtle, you should add **light, setsh "greenlight** to the procedure. Without this, the light will turn red when you start the game, if it was already green.

```
to play
  setup
  everyone [clickon]
  light, setsh "greenlight
end
```

Finishing Touches



- If the turtle is pointed down, it could end up going through the bottom of the page. It would then reappear at the top, reaching the sky color without crossing the road! To avoid this, add a thick line at the bottom of the page and program whatever color you use for the line to reset the turtle's heading to 0.
- Freeze the page to keep the player from dragging things around during the game.

Share with a friend!

Ask some friends to play your games and make suggestions for improved game play. Then, use MicroWorlds EX to publish your projects on the World Wide Web so anyone, anywhere can enjoy them. The following instructions will enable you to put your “Squish the Turtle” game on the web for millions to play and enjoy!

Note: If you do not have experience with posting files on the World Wide Web, ask a friend or your school’s network guru.

Choose Create HTML Template from the File menu. When creating an HTML template, MicroWorlds EX asks you to specify a location for saving the mxw and html files. These files must be saved in the same folder. On most operating systems, the dialog box allows you to browse and create a new folder for saving the files in. In some others (Windows 98), you must create the folder yourself, prior to this operation.

When saving a project as an HTML template, MicroWorlds EX creates two files with the same name and different extensions (for example.: MyProject.mxw and MyProject.html). Both these files and any media files must be saved in the same folder, whether you run them directly on your computer or on the web.

Double click on the html file to view your project in your Internet Browser. You will need to be sure that you have the latest MicroWorlds Web Player installed on your machine. You can download this for free from <http://www.microworlds.com>. You may also wish to make sure that your browser has enough RAM to run the project. On a Mac, set the amount of memory allocated for your browser for as much as you can spare, typically > 35 mb.



Challenges:

- Illustrate your screen to make it more attractive
- Design some new vehicle shapes
- Replace the navigation buttons with turtles. Design turtle shapes to look like the appropriate arrows.
- Figure out a way to get some vehicles to travel at half or double the speed
- Create a crash complete with sound effects and a realistic or humorous animation