Create Your Own Computer Games!

MicroWorlds Pro is a multimedia programming environment that offers you the possibility to create dynamic projects. You can use it to enhance your understanding of MicroWorlds and to get a real sense of the depth and breadth of this powerful multimedia programming environment (MicroWorlds Pro, 2000). Among other things, MicroWorlds Pro allows you to create animations and simulations, which you can easily turn into Games! Games are interactive projects and thus the process of their creation will help you learn more about programming and project development.

Degree of Difficulty

Experimental: Moderate
Conceptual: Moderate

Prerequisite Knowledge

This project is suggested for students that are already familiar with MicroWorlds Pro and Logo programming. In case you are not, please read the introductory part of Project 15 (“Create Your Own Simulations”) and complete all the activities of Part A before starting this project.

Objectives

Completion of the activities should enable you:

- to learn how to create animations and simulations by using MicroWorlds Pro.
- to use MicroWorlds Pro as a tool to create interactive games.


Useful Information: The Right Click on your PC’s mouse is used to open pop-up menus on objects; the same functionality is obtained on a Mac by Control Click. The PC’s Shift key is used to add more than one shape to a turtle and to change the heading of the turtle; the same functionality is obtained on a Mac with the Command key.

Do not forget to save your work regularly! A good idea is to save them as a series of project files, each at a different stage of development.
Each MicroWorld Pro (MWP) project consists of objects and text presented on separate pages. For every project you start, there is a corresponding single page. A page is accompanied by Menus, Toolbar, Status Bar, Command Center, and Tab Areas (see figure 2).

**Figure 2**

It is also important to know about Logo’s punctuation and marker meanings. According to MicroWorlds Pro (2000):

<table>
<thead>
<tr>
<th>Punctuation and Markers</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; Quotation mark</td>
<td>Indicates that the next word is to be taken literally.</td>
</tr>
<tr>
<td>[] Square brackets</td>
<td>Surround a list.</td>
</tr>
<tr>
<td>: Colon</td>
<td>Sometimes referred to as dots. Indicates that the next word is the name of a variable. Refers to contents of the container.</td>
</tr>
<tr>
<td>() Parentheses</td>
<td>Group things in ways that Logo ordinarily would not or varies the number of inputs for a primitive.</td>
</tr>
<tr>
<td>[ ] Vertical bars</td>
<td>Surround a long word so you can include spaces.</td>
</tr>
</tbody>
</table>
Procedure

After completing the introductory activities of Project 15, we are now ready to proceed into creating interactive games. Developing an interactive project like this one will help you learn about the power and the potential of computer programming. As Lough (2000) mentions, “you will be able to initiate and manage complex actions, and have some creative control at the same time.”

The developmental process of your game is also important to learn, because it not only challenges your programming skills in MicroWorld Pro, but also your knowledge about the game. It is important to know the rules of the game, otherwise you will not be able to make a fully functional game. A suggestion would be to outline the rules of your game, if it is a game that already exists, or define the rules of your game, if it is a game that you are creating, and try to match them with corresponding commands or programs before you start creating your MicroWorlds Pro page. After you finish, begin combining the programs that control the rules of your game into one program.

Let’s say you want to create a game called “Help the dog”, in which a dog has to move across highways without being hit by a vehicle. Since it is a new game that you are creating, a set of rules must be defined. For example, the rules of this game might be

- The dog must be able to move in any direction.
- There must be more than two lanes of vehicles passing through.
- The distance among the vehicles will not be the same.
- All vehicles in one will move in the same direction and with the same speed.
- There must be an option for changing the magnitude of the speed so the game can become easier or harder.
- There must be some sort of notifications that the dog has been hit by a vehicle (i.e., a crash sound) or that it has reached its destination.

Having in mind the aforementioned rules, we are now ready to select the main features that our game must include:

- There will be a dog that can move forwards, backwards, to the right, and to the left by using buttons.
- There will be four rows of vehicles (cars, motorcycles, trucks, etc.). Each row’s vehicles will be moving at the same direction and speed (the speed will be the same for all vehicles, but it will be changeable - it will be controlled through a slider). Some of the other rows’ vehicles will have different direction (opposite) and speed.
- The distance among the vehicles will be fixed but not the same. However, there must be enough space in every row of vehicles that allows the dog to pass through.
- There will be a notification that the dog was hit and a notification that you have completed the task of the game.
Finally, we have to decide what exactly the plot/background of the game will be. This is again a very serious issue and you have to consider it very carefully, based on the following criteria:

- The plot/background must clearly show the figure of your game. Avoid using in your background the colors that the dog shape consists of.
- The creation of the plot/background on your MicroWorlds Pro pages must be feasible. It is suggested that you use already existing shapes, icons, pictures, etc. (remember that you can import pictures in MicroWorlds Pro from other programs, the internet, etc.). Creating them from scratch is usually time consuming and sometimes impossible. It will be helpful if you have a drawing before starting.

Before starting, always, make sure you have a clear plan that includes a diagram of what you want to create or import, the program/vocabulary you will use, and the control equipment you will use (i.e., buttons, sliders, text boxes, etc.). After you specify all these desired parameters of your game, you can proceed with its creation.

For this project let’s consider developing the “Helping the dog” game described above (the idea for this game is based upon an already existing game, named frogger, developed by LCSI). The whole developmental procedure of the game is described in the following steps.

1. Create the background of your game by importing pictures or shapes from the Graphics area (you can also import pictures or shapes from elsewhere, i.e., internet), or by using the available painting tools of Graphics. Include two highways, and grass lanes. An example of such a background could be the following.
2. Create two turtles, turn the first one to a dog and the other one into a cat (you will find the shapes in the Graphics area), and position them, as follows.

3. Copy the cat’s image onto the background graphics. To freeze the cat at a certain place, select the stamper and click on a turtle. Stamping a turtle copies its image onto the background graphics.

4. Create two landmarks at the bottom of your page to define the starting point. Use for example the blue ball from the Graphics area. Of course, you can use any of the shapes that are available to you, just make sure that when you will be creating your page’s background these shapes do not overlap with your dog’s shape or your vehicles’ shapes. In case they overlap and you really want to include a particular shape, MicroWorlds pro offers you the option to minimize or maximize shapes. For example, the maximizing button of the Toolbar can have the following effect on a turtle.

In case you use the blue balls to define your starting point, place them at the bottom of your page as follows (the distance between the two balls is not of particular importance; it depends on your game rules and more specifically how difficult you want to make your game)
5. Create 10 turtles. Place five of them facing towards right and five of them facing towards left. You can orient the turtles towards right by using the command `seth 90` and towards left by using the command `seth 270` (see project 15, Part A, step 2 for more details). If you do not give them an orientation, they will move in the direction of their heading and this will mess up your game. Turn them into vehicles, and place them on the highway as follows.

6. Copy the already existing dog shape from the Graphics area, paste it to an empty shape area in the Graphics, and name it `deaddog` (the purpose of this is to have a
shape that shows what happens to the dog in case it gets hit by a car). Double click on your copied shape and from the pop-up dialog box click on the upside down flip option 🔄 and then on OK. Check your Graphics area to see that your copied shape is transformed to an upside down dog shape. Thus, in the scenario that the dog is hit by a car your page will look like

7. Name all of your turtles. Use the following names (start from the top of your page and move from left to right). In the pop-up dialog boxes, for all 10 vehicles, include in the instructions area the commands `fd carspeed wait 1`, and for the cat include in the instructions area the commands `if touching? "zach "vasso [zach, wait 2 win setpos [5 -150]]`, (see task 7 for more explanatory information). Do not include any commands in the dialog box of the dog.

<table>
<thead>
<tr>
<th>Shape</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>vasso</td>
</tr>
<tr>
<td>motorcycle</td>
<td>motor</td>
</tr>
<tr>
<td>blue car</td>
<td>car1</td>
</tr>
<tr>
<td>red car</td>
<td>car2</td>
</tr>
<tr>
<td>purple car</td>
<td>car3</td>
</tr>
<tr>
<td>yellow car</td>
<td>car4</td>
</tr>
<tr>
<td>red jeep</td>
<td>car5</td>
</tr>
<tr>
<td>red car</td>
<td>car6</td>
</tr>
<tr>
<td>yellow car</td>
<td>car7</td>
</tr>
<tr>
<td>orange tractor</td>
<td>car8</td>
</tr>
<tr>
<td>red car</td>
<td>car9</td>
</tr>
<tr>
<td>dead dog</td>
<td>deaddog</td>
</tr>
<tr>
<td>dog</td>
<td>zach</td>
</tr>
</tbody>
</table>
8. Create a slider. Type in its dialog box, `carspeed`, and set the minimum value to zero and the maximum value to 8.

9. Create five buttons. Name the first one `up`, the second one `down`, the third one `left`. (make sure to include the dot after left because the button’s name must be different from the logo command `left`), fourth one `right`, and fifth `start` (in all cases set the button to Once).

10. At this point your page should look similar to the following page,

![Page with buttons and slider](image)

11. Record three small pieces of sound or create three small pieces of melody (about 3-10 seconds each). You will need one introductory piece for starting the game (save it as `startmusic.wav`), a second piece for the vehicles hitting the dog (save it as `deaddog.wav`), and a third piece for winning the game (save it as `yeah.wav`). In case you want to create your own melody, select `Melodies` and follow the directions given in the help menu under the topic **MicroWorlds Objects: Melodies**. In case you want to record your own music or sounds, select `Recording Sounds` and follow the directions given in the help menu under the topic **Recording Sounds**. Both procedures are extremely easy, so try to use both (create a melody for one piece and record a sound for the other pieces). Note that after you create your sound files, you have to import them into your program (go under file, choose import, and then select sound). Check in the Project Tab area if the sound files were imported. If not, try again.
12. Now that the background, the turtles, the buttons, the slider, and the music files are set, it is time to start programming the game. The best way to do this is to break down our program in smaller parts and develop programs for each part separately. At the end you can combine all of them and create your game’s program. It is suggested to control your turtles’ actions with procedures where possible instead of placing commands into dialog boxes (Lough, 2000). During the development of your project, you will typically make several changes. It is usually easier and faster to edit procedures in the Procedures Tab area than to change commands in dialog boxes one at a time. This is especially important for this particular project because it includes a large number of turtles (Lough, 2000). The following tasks are the programming steps you have to follow to create the program of this game (the explanation notes are included for you to understand the reason behind selecting the particular commands):

**Task 1: Program the start button and all the related activities.**

In this introductory part of the program, you have to include the commands that control all the associated activities that relate to starting the game, such as, playing the `startmusic.wav` music file, positioning the dog at its starting point, making all vehicles start moving with controlled speeds based upon parts of a program you will develop for this purpose (in the introductory part you only name/introduce those other parts of the programs that control the behavior of your turtles i.e., `chkspeed` or `forever [check]`).

```plaintext
to start
startmusic.wav
chkspeed
everyone [clickon]
dog, setpos [5 -150]
forever [check]
end
```

**Explanatory notes:** This part of the program

- sets the game to start with music from the file `startmusic.wav` (as soon as you click on the start button of your game, the music will start playing),
- turns all the turtles on (everyone [clickon])
- indicates that the turtles will have a checked speed (chkspeed). Thus, a program must follow specifying how the turtles’ speed will be controlled. However, there must be another program explaining when the turtles must be checked.
- sets the dog to start from the point (5, -150). Note that the point (0,0) is the center of your page.
- indicates that all the turtles will be always under control (forever [check]) based upon a program named check that will be specifying how the turtles will be controlled.
Task 2: Define chkspeed.

It was mentioned in the introductory part (task 1) that the command \texttt{chkspeed} (or \texttt{checkspeed}) must be defined in order for the MicroWorlds software to understand what functions to perform when it reads \texttt{chkspeed}. Thus, you have to include a part in your program that specifies how the turtles’ speed will be controlled.

\begin{verbatim}
\texttt{to chkspeed}
\texttt{car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,}
\texttt{setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,}
\texttt{setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,}
\texttt{setspeed carspeed end}
\end{verbatim}

Explanatory notes: This part of the program

- controls the turtles’ speed (it is the definition of the program we were referring to, in task 1, with the command \texttt{chkspeed})
- sets the cars’ speed to be controlled by the slider named \texttt{carspeed}. For example the commands \texttt{car8, setspeed carspeed} mean that car8’s speed is set and controlled by the slider carspeed. Notice that for the vehicles of the second and third row we added a factor of 2 that makes those vehicles move slightly faster than the vehicles of the first and fourth row. The purpose is to make the game a little bit harder and more exciting. You can always exclude the +2 factor or change it (make it smaller or bigger).

Task 3: Program the up, down, right, and left movement buttons

One of the rules we set at the beginning, when we were designing the game, was to offer the possibility to the dog to move in four directions (up, down, left, right). The easiest way to include this particular feature in the game is to create interactive buttons. Each button must execute a series of commands, which in this particular case must be commands that define the direction and a specific range of distance the dog will move.

\begin{verbatim}
\texttt{to up}
\texttt{zach, seth 0 fd 47}
\texttt{end}
\texttt{to down}
\texttt{if xcor = 5 [stop]}
\texttt{zach, seth 0 bk 47}
\texttt{end}
\texttt{to right.}
\texttt{zach, seth 90 fd 25}
\texttt{end}
\texttt{to left.}
\end{verbatim}
zach, seth 90 bk 25
end

Explanatory notes: This part of the program

- sets how the dog must behave when the up, down, right, or left button is clicked. For example, to left. zach, seth 90 bk 25 end gives directions to the dog to turn 90 degrees and move backwards 25 units.
- forbids the dog to move backwards when it is on the x coordinate 5 (if xcor = 5 [stop]). This part was added, once again, to make the game harder. In case you move the dog upwards from the already set starting point (5, -150), you want be able to move it backwards, therefore you will have only three options to avoid the oncoming vehicles (up, left, or right). However, you can always move it first right or left, and then have all the options available. If you want to make the game more fun, include a number of similar restrictions in this part of the program.

Task 4: Define Check

It was mentioned in the introductory part (task 1) that, the command forever [check] must be defined in order for the MicroWorlds software to understand when the functions included in the part of the program named check must be performed. As the word forever indicates, the program check will always be applied. Now, you have to decide what to include in the check part of your program that you want regulated all of the time. For this project, we only need the speed of the cars to be controlled all the time. As for the rest of our turtles, the dog will be controlled by the buttons and the cat will be held fixed, therefore, none of these two have to be included in the check program.

to check
car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9, setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4, setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1, setspeed carspeed

Explanatory notes: This part of the program

- defines the duration that the turtles’ speed will be controlled (it is the definition of the program we were referring to, in task 1, with the command forever [check])
- sets the cars’ speed to be forever controlled by the slider named carspeed.
Task 5: Program dog’s reaction when hit by a car

One of the game’s rules is for it to end when a vehicle hits the dog. Consequently, a program that defines the dog’s behavior in the scenario that is being hit by a vehicle must be included.

```plaintext
if touching? "zach " car1 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car2 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car3 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car4 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car5 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car6 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car7 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car8 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car9 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car10 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
end
```

Explanatory notes: This part of the program

- defines a series of conditions that involves touching “zach” the dog. The conditions mention that if zach is touched by a vehicle (touching? "zach "car1), he will turn its image to deaddog’s upside down image (zach, setsh “deaddog) and after 2 seconds a program called dead will be activated (wait 2 dead). In addition, zach will go back to its already fixed starting point (5, -150), after the dead program has completed its functions, turning at the same time back to its initial upright image (we used number 1 for defining its initial upright image, even though it is named zach, because the program does not allow using zach twice within the brackets [ ]. This implies, though, that the shape of the dog must be placed at the very first spot in the Graphics area, or you can simply substitute 1 with the number of the spot the image is already in. (Click on your image once and you will see its position number in the Status bar).
**Task 6: Define dead**

The command **dead** that was included in the conditional statements of the previous part (task 5) must be defined in order for the MicroWorlds software to understand what functions to perform when it reads **dead**.

```plaintext
to dead
announce [Game over. Try again.]
deaddog.wav
end
```

**Explanatory notes:** This part of the program

- defines the functions that will be performed when the dead program starts running (it is the definition of the program we were referring to, in task 5, with the command **dead**)
- programs the game to pop up a dialog box announcing that the game is over and to try again, and play the sound file **deaddog.wav**.

**Task 7: Program finishing the game with a win**

In step 7, you were told to include in the cat’s instructions area the commands

```plaintext
if touching? "zach" "vasso [zach, wait 2 win setpos [5 -150]]. These commands, basically, initialize the win program (win) and then make the dog (zach) return to its starting point (setpos [5 -150]). The win program includes an announcement that informs you about your win (a pop-up dialog box will show you that **You Won!** – click on OK or ![OK icon] to continue with a new game), and a sound file (yeah.wav) that plays cheerful sounds or music indicating your win.
```

```plaintext
to win
announce [You Won!]
yeah.wav
end
```

**Explanatory notes:** This part of the program

- defines the functions that will be performed when the win program starts running (it is the definition of the program we were referring to, in the cat’s instructions area, with the command **win**)
- programs the game to pop up a dialog box announcing that “You Won!”, and play, at the same time, the sound file **yeah.wav**.

13. Put all the parts of your program together, as follows

```plaintext
to start
startmusic.wav
```
chkspeed
everyone [clickon]
dog, setpos [5 -150]
forever [check]
end

to chkspeed
car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,
setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,
setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,
setspeed carspeed
end
to up
zach, seth 0 fd 47
end
to down
if xcor = 5 [stop]
zach, seth 0 bk 47
end
to right.
zach, seth 90 fd 25
end
to left.
zach, seth 90 bk 25
end

to check
car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,
setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,
setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,
setspeed carspeed
if touching? " zach " car7 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car8 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car9 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
if touching? " zach " car10 [zach, setsh “deaddog wait 2 dead setpos [5 -150] setsh 1]
end

to dead
announce [Game over. Try again.]
deaddog.wav
end

to win
announce [You Won!]
cheers.wav
end

Copy the program and paste it into the Procedures Tab area.

14. Run the program, by clicking the start button. Vary the speed of the vehicles. What
do you observe? In case you want to quit playing, use the "stop" button of the Toolbar.

Activities

1. Find all the drawbacks regarding the design of the game described above and revise
the program to avoid them (hint: if you move the dog left or right and then move it
backwards, what happens?)

2. Include more restrictions to make the game more difficult. Make sure, though, that
passing the dog across the highways is possible.

3. Include a counter in your game to keep track of your wins and losses (hint: use the
information given in the help menu for counters).

4. Create a similar game, but this time, have the obstacles (i.e., stones) moving vertically
and the main character (i.e., a dragon) moving sideways.

5. Create a game that presents your main character (i.e., a soldier) jumping over moving
obstacles. Use similar rules as in the above game.

6. Create the musical game SIMON. Follow the instructions given in Lough’s book,
Learning MicroWorlds Pro (p. 51).
7. Follow a similar procedure, as in activity 6, and create a memory game that involves science knowledge. Instead of matching colors and music, as SIMON does, match questions with answers.

8. Expand the game described above by having the dog return back to its starting point in order to win.

9. Change the range of the slider and explain what will happen if you set the minimum value below zero. Define the range that you can use for playing the game.

10. Follow the instructions given in the Learning MicroWorlds Pro book (p. 73-75), and post one of your games on the World Wide Web (in case you do not have access to a server, run it on your hard disk).

**Final Project/Report**

Create a series of games that involve teaching a science topic/concept. Include a report explaining the reasoning behind your selection and an analytical step by step description of your games.

**References**
