Tutorial 4 : Randomness

A key element of many games is the concept of randomness. Many board games use dice to randomly choose a number that has an effect on the game. Card games also have an element of randomness to how they are dealt so that you get different results each time you play. This is what keeps the same games interesting to play even after you've already played them a couple times.

So how can we randomize our games using DarkBASIC? It's actually very easy. To show you, here is a sample dice rolling program.

```
print "Rolling dice..."
randomize timer()
wait 1000
result = rnd(5)+1
print result
wait 3000
```

Here's what I just did. The first and third lines you should already know about from previous tutorials. The 2nd line, you can think of as basically shaking up the dice. It gets the computer ready to use the command on the fourth line. The 4th line sets the variable `result` to a random number between 1 and 6. Here's how: when you call the rnd command, you have to give it a number in parentheses. Then, it will return some random number between 0 and the one you gave it. So, `rnd(5)` returns a random number between 0 and 5. But that's not the range I wanted. I wanted a range from 1-6. To fix this, I just added 1. Then, the least it can be is 1 (0+1) and the most it can be is 6 (5+1).

We can also use Math to get a negative range:

```
print (rnd(5)+1)*-1
wait 3000
```

Can you see what range of numbers this will give us? rnd returns some number between 0 and 5. Adding one gives us a range of 1 to 6. And finally, multiplying by minus one gives us a range of -1 to -6.

Before I show you a practical way to use this in a game, I want to tell you about one other group of commands that are extremely important. They are the loop commands.

There are basically three main types: do, while, and for. I'll try to explain each one with an example:

```
do
      i=i+1
      print i
      wait 1
loop
```

Try typing this in and running it. What will happen is you'll get a continual stream of numbers getting bigger and bigger until you finally cancel the program by pressing the Esc key. Why is that? As I mentioned before, programs run from top to bottom unless they are expressly told to do something different. This is what I did here.

When it got to the "do", the computer jumped over it, and went on to the next line. Here it added 1 to the variable i and on the next line printed it out. When it got to the loop command, though, instead of just continuing to run down the program, it went back up to the "do". As you can expect, it then added another 1 to i, printed it out, read the loop again, and again went back up to the top. This can go on indefinitely until either the user presses the Escape key or you give it an exit command. Here is another program I can write with the do loop:

```
do
      if i>=500 then exit
      i=i+1
      i=i*2
      print i
      wait 1
loop

print "That was quick"
wait 3000
```

Here I do the same thing except for one minor difference. In this case, I say that if i is greater than or equal to 500 (>= is greater than or equal to and <= is less than or equal to), the program should exit the loop. When it hits the exit command, it goes to the line immediately after the loop. In this case, it prints out "That was quick" and waits 3 seconds before ending.

In a case like the above example, there's a different kind of loop that's more suitable…the while loop:

```
while i<500
      i=i+1
      i=i*2
      print i
      wait 1
endwhile

print "That was quick"
wait 3000
```

This says, "While i is less than 500, do everything between this line and the endwhile." So, since new variables are automatically set to 0 when you first use them, when the computer gets to this the first time it sees that 0 is certainly less than 500 so it goes on to the next line. It then continues through the next three lines until it gets to the endwhile. At this point, it goes back to the beginning and again checks to see if i is less than or equal to 500. Finally, when i goes over 500, the while statement recognizes that and makes it skip down to the first line after the endwhile and it continues to run in top-down order.

Another loop that's helpful if you just want to repeat a certain number of times is the `for` loop.

```
for i=1 to 10
      print i
next i

print "Liftoff!!"
wait 3000
```

What this loop does is when the computer first reads it, it sets whatever variable you give it (in this case, i) to the first number after the = sign (in this case 1). It then does everything between it and the `next i` statement. When it reads the `next`, it adds 1 to the variable i and goes back to the beginning. It does this until i is greater than the maximum number at which point it jumps to the line after the next.

Before telling you about the game you can make using this information, I'd like to show you another command that can be helpful. In a previous example, the input command got the user’s input as a string. The problem with that is that you can't use a string to tell a for loop how many times to repeat. However, there is a way that you can convert that string into a number. Here’s how to do this:

```
input “How many times should I say Hello? ”, answer$
number=val(answer$)

for n=1 to number
     Print “Hello”
next n

wait 3000
```

The val command takes a string which contains a number (i.e. “4”) and converts it to that number. That enables us to use it for things like the for loop. It also turns out, that instead of doing this, you could simply change the `answer$` variable on the first line to just `answer` and it would input it directly as a number. Now I've covered enough information for you to make a number guessing game. Here’s how it works:

The computer chooses a random number from, let’s say, 1 to 10. The user tries to guess what that number is. If the user guesses incorrectly, it allows them to keep guessing again until they get it right. Then, it congratulates them. Here’s a basic framework for how to make this kind of game.

You make a variable called `answer` and set it to a random number between 1 and 10 (remember how to do this).

Then, you have a do loop that repeatedly does the following things:

First, it has an input statement asking the user to type in a guess. To simplify things, make the variable it stores their input in, just a regular number variable with no $ (e.g. `input “Please guess a number : ”, guess`)

Then, write an if statement which checks if their answer equals the random number chosen. If it does, print a congratulatory message, wait awhile, and end the program. Else, it prints an “Incorrect, please guess again.” statement. Then, it reads the loop command and goes right back to the beginning to allow them to guess again.