

ETOYS PONG TUTORIAL

(c) 2005, 2010 Lenny Pitt. Permission is granted to copy in whole or in part for nonprofit educational use.

ETOYS PONG TUTORIAL

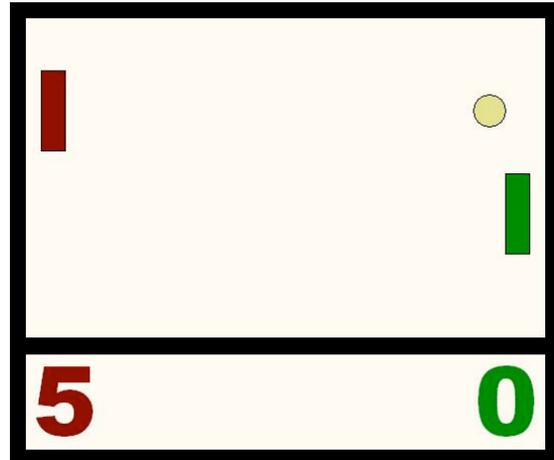
Overview

For our mini-project, we'll be making a simple Pong game

As with any decent sized project, we need to think about the design ahead of time and plan carefully. Doing this will save us a lot of confusion and work later.

We go through the following steps:

1. Brainstorming
 - a. Overall look and feel and behavior of game
 - b. General conception
2. Specify
 - a. Background graphics
 - b. User control, buttons, interface needed
 - c. Objects needed
 - d. How objects behave under what conditions
 - e. Flow of control
3. Tentative object and script design
 - a. What tests are needed for the behaviors of each object?
Do we know how to do it in Etoys?
 - b. What properties of objects will we need to change?
Examples: headings, coordinates, pens, bouncing, sounds.
 - c. Do we need some NEW properties that we define (user-defined variables)? Examples: health, score, lives
 - d. Draw a pictorial representation of the flow of control.
What scripts start? What scripts stop? Other scripts?
4. Create objects and scripts
 - a. Test as you go
 - b. If you have many objects all the same, then design only one first. After it is completely designed and working, then create copies and modify as needed. This saves a lot of work.
 - c. Re-evaluate overall approach as problems are discovered.



DESIGN:

1. Brainstorming

Let's define our Pong game:

Two player Pong game. A simple ball going back and forth, bouncing off of walls and can be hit by paddles controlled by users. After a point is scored, the ball is re-served. When one player gets enough points, he or she wins and the game is over.

2. Specify

a) Background graphics: none – simple blank playing field

b) User control, buttons, interface needed

- Button to start game
- Slider to control ball speed
- Slider to control paddle speed
- Score indicators
- Announcement of winner
- Instructions that can be hidden

c) Objects needed

- Ball
- Two paddles
- Score boxes
- Sliders
- Instruction text with "hide/show" button

d) How objects behave under what conditions

- Ball
 - A serve sets it in motion randomly towards 1 of 2 players
 - When it hits top or bottom wall, it bounces
 - When it hits paddle, it bounces
 - When it hits either back wall, it stops and re-serves
- Score boxes:
 - Scores start at 0
 - When the ball hits the back wall, then the appropriate player's score gets incremented by 1
 - When winning score is reached by either player, then game is over.

- Paddles:
 - Do they always move, with players changing the direction? (This is advised with two-player games so that players don't have a key-pressing fight)
 - Can they go forward and back or just up and down?
 - What controls their motion? (keys, mouse clicks...)

e) Flow of control

- Game start button
 - ❖ Resets score
 - ❖ Re-centers paddles
 - ❖ Resets ball and serves
 - ❖ Starts paddles moving
- Paddle moving
- Ball moving simultaneously
- Ball/paddle collision condition
 - ❖ ball changes direction
- Ball/wall collision condition
 - ❖ Stops ball motion script
 - ❖ Changes score
 - ❖ Checks for win
 - ❖ Resets ball
 - ❖ Re-serves
- Win condition
 - ❖ Stops all motion
 - ❖ Resets positions
 - ❖ Announces winner

3. Tentative object and script design

Playfield: A confined area to hold our game

Ball:

1. Simple round sketch
2. Serve script:
 - Set x, y coordinates to center
 - Set heading to random
 - Start ball motion going

3. Ball motion script
 - Forward by fixed amount
 - Test for bouncing against top/bottom
 - can use "bounce" command
 - Test for bouncing against paddles
 - test colorsees or overlaps object
 - how to change heading??
 - Test for hitting walls for scoring
 - How to implement? Two ideas: 1) Test x coordinate of the ball; if it's behind paddles, then the player scores. 2) Paint two walls and use "colorsees"

Paddles:

1. Simple rectangular boxes or sketches
2. Script for moving up and down
3. If it is a 1 player game, use mouse to control (but how does "computer" play?)
4. If it is a 2 player game, how do both give input? You will need to learn about keyboard interface.

Score boxes:

1. Where do the scores come from?
2. These are user-defined variables (properties).
3. It makes sense to assign them to the paddles, which represent the players.
4. Score boxes can be watchers.
5. Need a "scorepoint" script that credits the appropriate player.

On the next page you'll find an example of a script worksheet that you might use. Software designers use a variety of tools to help them organize their projects. This is essential when a program has hundreds of thousands, or even millions, of lines of code, and involves the coordination of many people. Even for small projects, a good design, well thought out and diagrammed, will help solve problems *before* they occur.



fire-once

_____ 's script _____



ticker

what it does:

properties of *this object* that it affects
(and how)

scripts that this script



fires



starts



pauses

IMPLEMENTATION

PLAYFIELD

The playfield is a special object taken from the supplies box. Playfields have their own coordinate system, and an object put inside it will have its x and y coordinates correspond to where it is in the playfield, not where it is in the world.

BALL

1. Draw a ball sketch, name it "ball" and drop it into playfield.
2. Make a **goHome** script that sends it to the middle of the playfield. To make this easier, we can set the playfield option so that the origin (0, 0) is at the center. This is done by alt-clicking on the playfield, selecting the black and white icon from the halo, choosing the "playfield options" item at the bottom, and clicking in the box that says "origin-at-center". (This option can also be chosen for the world as well.)



3. Make a basic motion script for the ball that moves forward and bounces off walls. Perhaps adjust the amount forward and the tick rate.



4. Make a serve script that first sends the ball home, then sets the ball in motion in a random direction.



Now is a good time to test whether this is what we want. Do you notice any problems? We may not want the ball to head "random 360", since a serve that is more vertical than horizontal will take a long time to get anywhere. We can fix this later, as well as put in sliders for the ball's speed.

BOUNCING OFF THE PADDLES

There were two more ball behaviors to worry about. One was how to tell when a point has been scored; the other is bouncing off the paddles.

Let's first take care of the paddles.

Drag a rectangle out of the Supplies box, resize it to look like a pong paddle (long and thin), and rename it "leftPlayer". Place it in the playfield near the left edge. Recolor it however you like, with a single color.

Now, the geometry challenge question: If a ball hits a vertical surface while heading X degrees, how many degrees is it heading after it rebounds?

Some examples:

- Initial heading: 45 new heading: 315 = -45
- Initial heading: 90 new heading: 270 = -90
- Initial heading: 210 new heading: 150 = -210

This suggests that we can modify the heading of the ball after it hits the paddle, either by subtracting its current heading from 360 or simply by multiplying by -1.

We now need a test for the ball touching that paddle. Get a "TEST" tile (for example, from the tests pane in the viewer), and then create the following:



The "ball's color sees" test is found on the test pane for the ball. When you drag it out, you will get two red color boxes, like this:



You then click in each red box to set the actual colors you want tested.

In this case, the first box is set to the ball's blue color (by clicking the eyedropper that appears on the blue ball), and the second box is set to the particular red color of the paddle, which is perhaps a slightly different red than the one that initially appears.

Now place the test in the ball's move script and see what happens when the ball hits the paddle.

Test it using different ball speeds!

Problem 1

If you play a bit, you may find that sometimes the ball gets stuck on the paddle. Can you figure out why?

The answer is that since the ball is moving several steps at once, sometimes the first time it touches the paddle, it is overlapping the paddle by several pixels. (That is, it is not just touching edges.) Then when the ball changes direction and moves a little, it may still be overlapping, so it changes directions again at the next tick.

How can this be avoided? Try adding another command under the "yes" condition that the ball should move forward some small amount (5 or 10) to automatically get it off of the paddle. This command should be placed *after* the heading change.

The ball move script should now look like this:



Problem 2

If the ball is moving fast enough by virtue of a command that says "forward 30" or "forward 50" or forward some reasonably large number (anything wider than the paddle and ball), it may actually step completely over the paddle without touching it. The way to correct this is to either make a thicker paddle or ball, and/or to achieve the desired speed by increasing the ticking rate of the move script and decreasing the distance moved by the ball at each tick.

PADDLE Movement

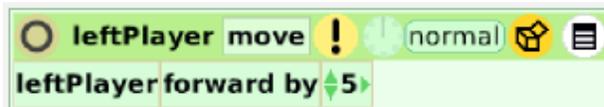
There are a number of design considerations here. Recognizing that there is only one mouse, we could choose to have: 1) one player uses the keyboard, and the other uses the mouse, or 2) each player uses the keyboard, to move their paddles.

Experience suggests that having both players trying to hit their input a lot and often will cause trouble; one player may hold down his or her key, or hit it so many times, that the other's input does not get noticed. A good design for this system is one where not many keystrokes per player is needed, and we rely on good behavior of players to not hit keys endlessly to cause the other player troubles.

Thus, the paddles should either move up and down in fairly large increments each time a key is hit, or they should always be moving and the keystrokes only serve to change direction. This way, there is little advantage to many repeated keystrokes.

Let's choose the "always moving" approach, which gives the users some additional challenge, since they will not be able to stop their paddle in one place.

We will want basic motion commands for the paddles that go forward by a fixed number at each step, just like the ball's move script:



In the viewer for leftPlayer, open up a pane for the "input" category. You'll see "leftPlayer's last keystroke". Try hitting a few keys and see how this changes. The last keystroke tells the program what key was last hit. We can use this in a test to make the paddle change directions.

Assume we want leftPlayer to be controlled by "a" and "z" for "up" and "down" directions.

Here is the start of the leftPlayer move script. Create one like this, but with one more test (for the case that the input key hit is "z", in which case you'll want the heading to change to 180 (i.e., down)). .



Test your script by setting it ticking, and making sure that the "a" and "z" keys indeed make the paddle change directions up and down. You may want to change the ticking rate, or the amount that the paddle moves forward at each tick.

NOW is the time to create the rightPaddle. Bring up the halo of the left paddle, and click on the "copy" icon in the upper right corner. (DO NOT hold down "shift" while clicking – that is for a different technique which may be covered in another tutorial, and is not appropriate here.)

Open the viewer for the new paddle. Rename it rightPaddle. Change the keys in the last keystroke test to "k" and "m" so that these keys control the direction of the paddle. The move script for rightPaddle should look identical to the one for leftPaddle with the exception of these two small changes. Now test the new paddle. It should move up and down and change directions by hitting "k" and "m". Check that the ball bounces off of it as well.

SCORING

A. SETTING UP SCORE VARIABLES AND WATCHERS

We need a way to keep track of the score of each player and to change the score when the ball sneaks by a paddle.

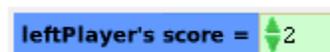
Recall from earlier lessons that every object created has built-in variables which we've called "properties" (such as *x*, *y*, *heading*, *pendown*). We can also create our own new variables for an object, to store information that we'd like. The new variables can have different value types, just as the properties can have value type of number, color, Boolean, string, etc. For score, we'd like a number. It makes sense to give to each player a variable called *score*. The players are represented by the paddles *leftPlayer* and *rightPlayer*.

1. Create a variable *score* for the *leftPlayer* by clicking on the "v" on the top of its viewer. By default, the value type for a new variable is "number", which is what we want in this case, so we do not need to change it. Make sure to name it and not leave it as "var1". When you're done, a pane that looks like this should appear in the viewer:



2. Click on the small menu icon next to *leftPlayer* in the line above and select *decimal places* and set it to 0, since we'll only be using whole numbers.

Again, click on the small menu icons and this time click on *detailed watcher*.



Place the tile wherever you would like. This will keep the *leftPlayer's* score.

3. Repeat steps 1 and 2 for the *rightPlayer*.

B. SCRIPTING THE CHANGE-OF-SCORE CONDITIONS

We need a way to test when the ball has gone beyond a paddle. There are several ways to do this:

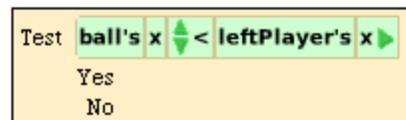
- We could check if the ball's x coordinate is smaller than or greater than a certain amount, indicating that it has gone too far to the left, or too far to the right, which means somebody should get a point.
- We could draw a thin color strip along the left and right edges, and if the ball hits that color we know it has gone beyond the paddles.

One way of implementing the first option would be to figure out how far to the left the ball can be just before it hits the left wall. Try dragging the ball very close to the left edge, but not quite touching. Then look at the ball's x-coordinate in its viewer. I get something like -310. We could then have

```
TEST    ball's x < -310
      Yes... (score a point for the rightPlayer...)
```

However, this might not end up working on other screens, or when the playfield was resized.

A better option would be

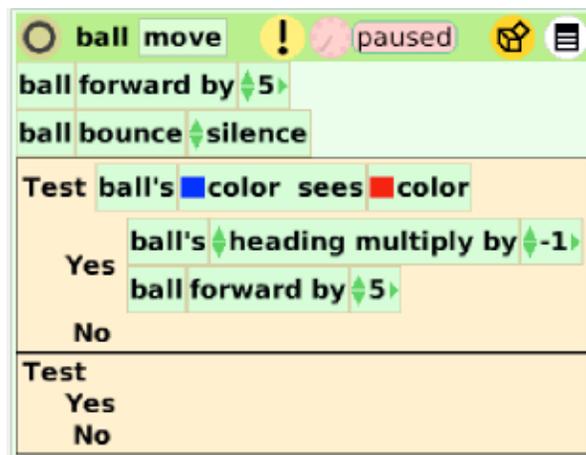


The second approach involves drawing a colored goal area down each edge. We can use a different color for the left and right edges. If instead of painting on the playfield, if we place down a colored object (for example, a thin rectangle of color) then we get the option of making this rectangle get shorter as the game goes on, making a smaller "target" area of color that needs to be hit. This technique is also useful if we want to make a game with a goal like soccer or hockey: the goal is colored a different color.

A final thought regarding these options goes back to a point mentioned earlier. If the ball is moving too much at each step, it is liable to step completely over the color strip, and no point will be scored. In this case, because we have the ball set to "bounce" within its move script, the only harm done will be that a point would not be scored, and the ball would just bounce back off of the wall.

We implement the goal area approach.

1. Drag out a blue rectangle from the Supplies box.
2. Name it "leftGoal", resize it to be thin and match the height of the playfield, and then place it along the left edge just inside the playfield.
3. Instead of a "colorsees" test as we did when testing the ball and paddle, we'll show a different method so that you learn more. Go to the ball's move script, and add a new "TEST" box below the existing one. Make sure that it goes below, and not inside of, the other test. You should have this:



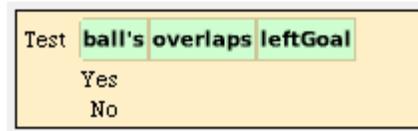
4. Open up the tests pane for the ball, and drag the tile and place it next to the "TEST".

ball's overlaps dot

This test is used to tell if one object overlaps another. We will replace the word "dot" with a tile for the leftGoal object.

5. In the halo for leftGoal, click on the orange tile icon (lower left), and place the tile over the "dot" tile at the end.

Your test should now look something like this:

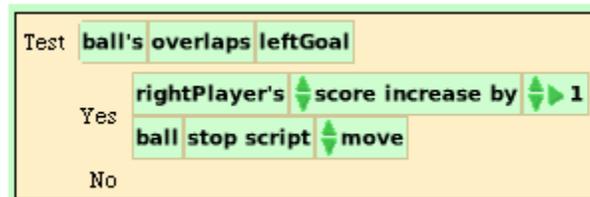


Now, what should the "Yes" event be if the ball does overlap the leftGoal? The rightPlayer should score a point.

- Open up a viewer for rightPlayer, look at the "variables" pane, and drag the green/white arrow, placing the resulting tile into the "yes" position in the test. Now click on the part that shows



And change it to "score increases by". Finally, add a "ball stop script move" tile, so that your entire test now looks like this (



- Repeat steps 1 through 6 with rightGoal and leftPlayer's score.

When a point is scored, changing the score is not the only thing that needs to happen. The ball must stop, and then be re-served, perhaps in a direction that depends on which player just scored, or perhaps in a random direction.

We'll leave fancy re-serving for an extension activity. Let's try to finish up this project as quickly as we can.

Just after a point is scored, we must test to see if that player has won. If so, everything should stop. If not, the ball should be re-served. Modify the two tests you created on the previous page of this tutorial as follows:



Notice that we have a test nested inside of a test.

We don't fill in the "Yes" condition, because we need to first create a script that ends the game gracefully, making sure that everything has stopped, the ball is reset, and perhaps an announcement is made about who won. We will return to this shortly.

BEGIN AND END

What should we make sure happens at the end of the game? What should we make sure happens at the beginning of the game? How do all of the different scripts interact?

BEGINNING OF GAME

At the beginning, we should be sure that all objects are where we want them, that all values are set to initial values (for example, scores should be 0), and that all scripts that should start ticking actually begin ticking.

1. Create a "newGame" script that belongs to the playfield Pong. The newGame script should do the following:



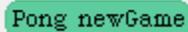
The scores are reset, the paddle move scripts start ticking (you can find these tiles on the "scripting" pane (not the "script" pane) for the paddles. You'll need to pull out "start script emptyscript", and change the "emptyscript" to the "move" script.

Note: Why are we "start"ing scripts move and paddlemove, but only "do"ing script serve? Shouldn't we also start ball move?

The difference between "start" and "do" is that "start" is like clicking on the clock icon, and "do" is like clicking on the exclamation point. "Start" starts the script ticking over and over, and "do" makes it fire only once.

We only want the script "serve" to execute once, because it puts the ball in the home position, and then IT starts the ball move script ticking.

2. Next, you can obtain a button that fires the Pong newGame script (it shouldn't start it ticking; it should execute once). To get the button, click on the black/white icon at the top right of the Pong newGame script header, and scroll down and choose "button to fire this script". You'll obtain a button like this:



Because it is a button, it resists being picked up once placed down, so put it where you want it, or move it later by using the black tongs icon from the button's halo.

Note also that this type of button *fires a script once*. It does not start a script ticking. It is useful for actions you want to happen once through. If you want to create a button that starts some script "move" ticking, you need to create a script (call it "startmove") that contains only the command "start script move". Then, get a button to fire script "startmove" exactly once.

END OF GAME

3. Create a script called "Pong endgame".

At the very least, it should make sure that all ticking scripts have stopped. These are leftPlayer, rightPlayer, and ball "move" script, and world's "paddlemove" script.

Find the tiles

```
"leftPlayer stop script move"  
"rightPlayer stop script move"  
"ball stop script move"
```

and place them all in the Pong endgame script.

4. Finally, get a tile "Pong do endgame" from the scripting pane for Pong, and insert it in the "yes" consequence for the tests you created earlier that are in ball's move script, that test whether the player's scores have reached 10.

Here are all of the scripts that you should have created so far:

ball move (paused)

```

ball forward by 5
ball bounce silence

Test ball's color sees color
  Yes ball's heading multiply by -1
  ball forward by 5
  No

Test ball's overlaps leftGoal
  rightPlayer's score increase by 1
  ball stop script move
  Yes Test rightPlayer's score = 10
    Yes Pong do end game
    No ball serve
  No

Test ball's overlaps rightGoal
  leftPlayer's score increase by 1
  ball stop script move
  Yes Test leftPlayer's score = 10
    Yes Pong do end game
    No ball serve
  No
    
```

leftPlayer move (paused)

```

leftPlayer forward by 5

Test leftPlayer's last keystroke = a
  Yes leftPlayer's heading ← 0
  No

Test leftPlayer's last keystroke = z
  Yes leftPlayer's heading ← 180
  No
    
```

rightPlayer move (paused)

```

rightPlayer forward by 5

Test rightPlayer's last keystroke = a
  Yes rightPlayer's heading ← 0
  No

Test rightPlayer's last keystroke = z
  Yes rightPlayer's heading ← 180
  No
    
```

ball goHome (normal)

```

ball's x ← 0
ball's y ← 0
    
```

ball serve (normal)

```

ball goHome
ball's heading ← random(360)
ball start script move
    
```

Pong newGame (normal)

```

leftPlayer's score ← 0
rightPlayer's score ← 0
leftPlayer start script move
rightPlayer start script move
ball do serve
    
```

Pong endGame (normal)

```

rightPlayer stop script move
leftPlayer stop script move
ball stop script move
    
```

BUGS

Often when writing programs, things do not happen just the way we expected. Some problems occur because there are many different things happening at once, and we failed to think ahead of how these will interact. These are hard to predict.

Other more common problems can be avoided with careful planning and thought. Typical mistakes to watch out for.

- Starting a script ticking instead of firing it once
- Firing a script once instead of starting it ticking
- Failing to stop a script when you should
- Making sure you are testing the right things, and that the tests conditions are "properly seated" next to the "TEST" statement

When things do go wrong, the way you can figure out what the problem is, is by carefully stepping through each script yourself, command by command, to see what it does. Often this reveals the issue. Step through one script completely, then the next, then the next, to see what happens during a single tick of the clock.

If you cannot figure out what is wrong, ask your instructor, or try some experiments.

EXTENSIONS

There are many ways you might improve or add to the pong game. Here are some we thought of. Some are easy, some are challenging.

- You may want to change the serve angle to be more acute, since it is an annoyance.
- Add an announcement about who won, using show/hide of the text.
- The ball speed increase as the score increases.
- The paddle size shrink as the score increases.
- The goal areas shrink as the score increases.
- Multiple balls appear as the score increases.
- Allow paddles to move forward and back
- Allow user-settable speeds for objects via sliders
- User-settable game points needed to win
- Make the angle of reflection of the ball off of the paddle depend on where the paddle hits the ball. Paddles should

behave somewhat as if they're semi-circles. Basic idea: current reflection should be linearly adjusted based on difference of ball's y-coordinate and paddle's y-coordinate.

- You may want to tune speeds to a particular machine. This is easiest to do if you use variables and have a single script that sets all of these just the way you'd like. Talk to the instructor about this if you are interested.
- The ball changes color based on who hit it last (need to replace colorsees tests with overlaps tests or ball/paddle interaction).
- Have "powerups" on the screen, which if hit, gives some special power to the person who last hit it (indicated by the color of the ball). Example powerups include immediate decrease of opponent's score. Or, add to some variable "power" for the player that allows a "turbo" boost in the ball's speed when they push a button, provided they have enough power.

What extensions can *you* think of and program?