# **Build your own PONG!**

## **Introduction**

Pong is one of the first electronic games created, based on the real-life game of table tennis. There are two players, each represented by a paddle on a rectangular screen. Each player controls their paddle by moving it up or down to hit the ball across the screen. If a player misses the ball with their paddle, the other player scores a point.

So the basic rules of the game are:

1. There is a background that shows the game board
2. There are two paddles on the left and right sides that move up and down
3. There is a ball that moves across the screen and bounces off the paddles
4. When a ball falls off the screen on the left or right sides (the player missed the ball), the other player scores a point.
5. There is a score that represents the points each player scored.

You can play an online version of this game [here](http://www.ponggame.org) to get a better intuition for how the game works. In this lab, you will be building your own version of Pong!

We provide you an outline of code for you to start this lab, but you will be writing most of the code that creates this game! This code skeleton will provide you many constants that will make your life easier, so be sure to use these constants in your code! We also take care of key handling for you, and you can read more about this in the *Key Handling* section below.

In addition to this README file, you will also find many instructions and hints in the code we give you! Be sure to **read the comments** carefully to determine what needs to be done inside each function!

## **Set Up**

You will be completing this lab in **Processing** and start with the 5 skeleton files we provide you inside the pong folder. These files are:

**FILES**

* **pong.pde**: This is the main game file that you will be editing.
* **constants.pde**: This file contains constants that will be useful to you! Make sure to read through this file before you begin, and you may need to
* **flow.pde**: This is a simple file that controls the flow of the game. You will update this code based on what should happen when a point is scored.
* **key\_handling.pde**: This file is for your convenience and does not need to be edited! It takes care of key handling.
* **physics.pde**: In this file, you will write code that determines how the ball's position and velocity will be updated as well as the position of the paddle using the physics you learned in class!

**Note:** **".pde"** at the end of the file indicates that the file should be **opened using Processing**.

### 

### 

### 

### 

### **INSTRUCTIONS**

**pong.pde** Just like any main Processing file, the functions that will be run are void setup() and void draw(). These functions, however, will mainly call other functions that you write

**FUNCTION:** void setup() is provided to you, which calls resetGame() to start a new game. **You** will write what happens inside resetGame() at the bottom of the file. Remember that setup() only gets called ONCE at the start of the program.

**FUNCTION:** void draw()is given to you, but simply calls drawGameScreen() which you will be writing. Remember that draw() gets called many times per second, and this is how the game screen will be refreshed each time something changes.

**constants.pde** This file consists of all sorts of useful variables that you will use throughout your code. Never use arbitrary values in your code; create a variable for it in this file if it doesn't already exist and then use that variable in your code. For example:

* DO NOT write size(500, 500)
* Instead, write: size(displayWidth, displayHeight);

Most constants in this file are self-explanatory based on the name of the variable and the comments throughout the file. If you still aren't sure what one of the variables is, ASK! Or look for where that variable is being used and make an educated guess. Here are some variables that may be confusing:

BALL\_VELOCITY: This is the maximum speed of the ball in each direction. In this lab, this variable will be set to the value 20. Then ballVx will be either -20 or +20 and ballVy will be anything between -20 and +20.

* For example: ballVx = 20 and ballVy = 20 means the ball is going at a 45 degree angle (down-right). If ballVx = -20 and ballVy = 10, the ball will be going down-left.
* Note the speed of the ball in the x-direction will always be the same, but will be changing in the y-direction.
* Note you should never be using the value 20 directly; always use BALL\_VELOCITY to represent this value.

## **Step 1: Create a ball and its properties**

The ball will be represented as a circle with the radius specified in the constants file. The ball also has a position and velocity throughout the game. These variables have been declared in the constants file with the appropriate names, but you need to set their values when you start a new game in the resetGame() function.

Write the function drawBall() to display the ball in its current position with the current velocity using the variables in constants.pde.

## **Step 2: Ball-Wall Collision Detection**

When the ball collides with any of the four walls or either of the paddles, some action must be taken:

* (DO NOW) -- If colliding with any of the four walls, then bounces off, just like the Movement Lab.
* (DO LATER in Step 7) If colliding with left or right padde, then ball bounces off

Later, we will modify this code so that the ball hitting the left or right walls will make the player score a point instead of the ball bouncing off, but for now, let the ball bounce from every side!

In the updateBallVelocity() function, write code to make the ball bounce from all four walls!

## **Step 3: Draw the paddles**

Now try to write your code for the drawPaddles() function to display the left and right paddles (rectangles) in the correct positions. To do this you will use the following paddle parameters defined in the constants file:

paddleLength: the height of the paddle paddleWidth: the width of the paddle paddleColor: the color of the paddle leftPaddle: the Y-position of the left paddle rightPaddle: the Y-position of the right paddle

Note that the X-positions of the paddles are fixed, and they should be based on the given variables (paddleWidth and width). Remember that you can draw a rectangle by specifying its center coordinates in the following way.

rectMode(CENTER);  
rect(centerX, centerY, rectWidth, rectLength);

## 

## 

## **Step 4: Move one paddle**

Fill out the updatePaddlePositions() function based on the keys that are pressed. See the key\_handling.pde file or constants.pde file to figure out what the different variables do.

Note that the paddles only move in the vertical direction (update the y-position of the paddle). By default, you should be using the following keys, to move the left and right paddles.

char LEFT\_UP = 'q', LEFT\_DOWN = 'a';  
 char RIGHT\_UP = 'o', RIGHT\_DOWN = 'l';

Feel free to change this if you want

## **Step 5: Restrict paddle movement**

Currently, the paddle can be moved off the screen; that's not good! Let's fix that. Modify your current code for updatePaddlePositions() to restrict motion of the paddles to make sure the paddles don't leave the screen. Once the paddle reaches the edge of the screen, it should not be able to move further in that direction.

## **Step 6: Move both paddles!**

To make things actually exciting now, make both paddles move independently. Edit your code in updatePaddlePositions() to make both the paddles move based on which keys are pressed!

## **Step 7: Ball-Paddle Collision Detection**

Again now you want to write your code to detect collision of the ball with the paddle. To do this you must write code that detects a large difference between the postion of the ball and the paddle as a miss, and a small difference as a collision.

Note that there are two conditions that need to be checked. For example, if we are checking collision with th left paddle.

* The *x*-coordinate of the ball should be small (close to the left edge). For instance, we could check whether ballX < paddleWidth + ballRadius, or something similar.
* The *y*-coordinate of the ball should be within the range of the left paddle. For instance, we could check whether ballY < leftPaddle + paddleLength/2, and ballY > leftPaddle - paddleLength/2.

## 

## 

## **Step 8: Ball-Paddle Reflection (Bouncing)**

If the ball collides with the paddle, the ball should be reflected. The simplest way of performing reflection, is to make the x-velocity of the ball negative i.e. ballVx \*= -1.

If you actually do this however, you may notice that the game isn't very interesting, since the ball is effectively just bouncing around a box, with no other velocity changes. We should also modify the y-velocity of the ball, *depending on the collision location*.

We will now imagine that the paddle is actually curved. So, if the ball hits the top edge of the paddle, it will be deflected upwards more. Similarly, if it hits the bottom edge of the paddle, it will be deflected further down. More precisely, if the ball hits the (top or bottom) edge of the paddle, it's y-velocity will be -BALL\_VELOCITY or +BALL\_VELOCITY respectively. A hit in the middle will result in some intermediate velocity.

Here is one possible formula you can use to compute the y-velocity after reflection from the paddle.

buffer = paddleLength/2  
BALL\_VELOCITY\*(ballY-paddlePosition)/buffer

## **Step 9: Player scores a point**

We will modify the code from Step 2 so that the ball colliding with the left or right wall will NOT make it bounce anymore. Instead, if it collides with the left or right wall, the opposite player scores a point.

We can detect this at the same time that we detect the collision of the ball with the paddle. As an example, if the first condition in **Step 7** is satisfied, but the second one is not satisfied, then the *left player* has failed to hit the ball with the paddle, and so the right player should gain one point.

To do this write your code in the rightLose() and leftLose() functions in flow.pde. Think about what happens when a player scores a point. Make sure to update the player scores and the game screen (**Step 10**).

## **Step 10: Display scores**

Write your code for the displayScores() function to display player scores on the screen. Processing contains a function called text: text(textValue, xPosition, yPosition)

This displays the textValue at the corresponding position. For instance, the line text("Hello", 100, 200) will display the text **Hello** at the position (100, 200) You may use this function to display the scores.

## 

## **Step 11: Create a start screen (Optional)**

Have a start screen that will be displayed when the program is first run. If the reset key is pressed or if either player scores a point, the game screen must return to this start screen as well.

One way to achieve this, is to use a variable int gameState = 0. In general, we can specify a different behavior for each possibe value of gameState.

* If gameState == 0, then we should show the starting screen. If the start button is pressed, then we can change the value to 1.
* If gameState == 1, then we should show the actual game screen. If the reset button is pressed, then we can change the value to 0 (to go back to the start screen).

## **Step 12: Pausing the game (Optional)**

If the game is paused, the ball and paddle properties should not be updated until the game is unpaused. Also indicate on the game screen that the game is paused. You may want to use text() function here again to display that the game is paused. You may also want to have a variable that keeps track of the game state (paused or playing) and update your other code until now accordingly!

## **Key Handling**

You are given defined keyboard keys that the player will use in the game. They will do their assigned functions only when pressed.

* In order to control the Left Paddle, the following keys can be used: LEFT\_UP = 'q' LEFT\_DOWN = 'a'
* Controls for the Right Paddle RIGHT\_UP = 'o' RIGHT\_DOWN = 'l'
* Game Controls RESET = 'r' PAUSE = 'p' START = 's'