# Processing Assignment 3: Adding the Cannon

Assignment Due 11/4

## Description

This deliverable will include several new features over the previous assignment:

1. Drawing the cannon. The most basic version will be a narrow rectangle.
2. Create a mechanism for adjusting the cannon’s angle. I would prefer the use of two keys that move the angle in a + or – direction. In this example those keys are < and >.
3. The angle must be kept between 0 (horizontal) and 90 (vertical) degrees.
4. The cannon angle must be displayed in the window.
5. There must be a fire button that launches the projectile. I suggest using the space bar.

## 

## Psudocode

### Changes to the draw() function

The draw function has been simplified by moving the calculation of xpos and ypos to the functions that use it. We still use the expression (currT > maxT) to determine when the cannon ball has completed its flight. Notice in the following ‘if’ statement that we only increment currT while it is less than maxT. When we stop incrementing currT, the ball stops moving on the screen. We also added a new function drawCannon() which will be described below.

void draw() {

background(255);

if (**currT > maxT**) {

checkTarget(); // Does nothing in this version

}

else {

currT = currT + deltaT;

}

**drawCannon**();

drawPosition();

drawShot();

}

### Rotating and Drawing the Cannon

Drawing the cannon involves performing rotating a rectangle around the end nearest the lower left corner. In Processing, this involves the following operations.

The steps of rotating and drawing the cannon are to 1) move the screen’s origin to the lower left corner 2) rotate new shapes 45 degrees 3) draw the rectangle with the new origin and rotation 4) restore the default shape settings.

The pushMatrix() function records the shape translations that follow. The translate() function moves the screen’s origin to almost the bottom left-hand corner of the window. The rotate() function rotates any newly drawn shapes the around the screen’s new origin. The amount of rotation is determined by the variable cannonAngle. The angle is converted to a negative to rotate CCW. The rect() function draws a rectangle using the new origin and rotation. The popMatrix() removes these translations returning Processing to its default drawing settings.

drawCannon() {

// Start new shape translations

**pushMatrix**();

fill(0);

// Move screen origin to the lower left hand corner

**translate**(0, height-5);

// Rotate new shapes cannonAngle degrees counter-clockwise

**rotate**(cannonAngle \* -1);

// Draw rectangle with the new origin and rotation

rect(0, 0, 40, 5);

// reset shape translations

**popMatrix**();

}

### Specifying the Cannon Angle

This version of the game allows the user to perform three actions through the keyboard 1) the angle of the cannon can be increased (raised) 2) the angle of the cannon can be decreased (lowered) 3) the cannon can be fired.

This code makes use of the Processing function ‘keyPressed()’. This function is activated (called) each time the user presses a key. The built-in variable key contains the value of the key that was pressed. Using the key value, the program can determine what action the user has taken. This determination is carried out in the nested if statements shown next. If the key value equals ‘<’, the user wishes to raise the cannon. If the key value equals ‘>’, the user wishes to lower the cannon. Finally, if the user presses the space bar (key == ‘ ‘) the user wishes to fire the cannon.

void **keyPressed**() {

if(**key** == '<') {

cannonAngle = cannonAngle + .01;

}

else if(**key** == '>') {

cannonAngle = cannonAngle - .01;

}

else if(**key** == ' ') {

fireCannon();

}

}

In order to keep the angle between 0 and 90 degrees, we need to verify the current angle in the appropriate if statement. For example, if we add the following highlighted code to the raise cannon option, we will always keep the angle at no more that .5PI radians or 90 degrees. Keeping the lower cannon angle greater than zero is left as an exercise to the reader (I’ve always wanted to say that). The variable HALF\_PI is built into Processing.

if(key == '<') {

cannonAngle = cannonAngle + .01;

**if(cannonAngle > HALF\_PI) {**

**cannonAngle = HALF\_PI;**

**}**

}

### Firing the Cannon

When we fire the cannon, we are restarting the simulation from currT (current time) equals zero. The following code demonstrates that the fireCannon() function does just this. It sets the current time to zero. It transfers the current cannon angle to the firing angle. It then calculates a new maxT with the new firing angle. Resetting currT to zero will restart the ballistic calculation (see the draw() function).

void fireCannon() {

currT = 0;

firingAngle = cannonAngle;

maxT = 2\*v\*sin(firingAngle)/g;

}

### Misc Changes

We split the status text into three lines and added the cannon’s angle.

We now maintain two angles: The cannon’s angle (cannonAngle) and the angle at which the cannon is fired (firingAngle). We maintain separate angles so that the user can adjust the cannon while the ball is in flight. We transfer the cannonAngle to firingAngle when the user fires the cannon.

## Deliverables

Each team will submit their sketchbook directory containing their submission on a USB thumb drive. Just copy the processing directory containing your work onto a USB thumb drive.

Also include a README.txt file on the USB drive containing your section and team numbers.   
**No File..No Grade.**

THE THUMB DRIVE MUST CONTAIN THE FOLDER WHICH IS THE PROCESSING SKETCHBOOK TO BE GRADED AND THE README FILE

Graders will evaluate the submissions by executing the submitted program file and ensuring that it executes correctly.