# Processing Assignment #3 “Crash” CMSC 101 / IS 101Y – Fall 2013

*Revised version! See ~~strikethrough~~ for deletions and red text for additions.*

## Due Date

* **Processing Assignment 3: Due Thursday, October 10**
  + ***Note:*** *The late policy for Processing assignments applies to this assignments (it may be submitted up to one week late, with a 15% penalty per day or fraction of a day late). However, you are* ***strongly*** *encouraged to start early and complete the assignment on time, so that you are well prepared to start the group project!*

## Collaboration Policy

* This assignment ***may be completed individually or in pairs.***You may discuss the requirements with students outside of your pair, but you must write your own code, and you may not share your code with any other students, search for code on the Internet, or use solution material that was created by anyone but yourself. You may not work with more than one other person, and you will both be responsible for understanding and being able to explain your program.

## Assignment: Requirements

In this assignment, you will implement a simple game in which the user can “launch” objects that will bounce around the screen until they bump into each other and explode. The easiest way to understand what you are supposed to produce is to view a movie of my implementation of the program:

http://www.csee.umbc.edu/courses/undergraduate/CMSC101/Fall2013/Handouts/movie.mpeg

Here’s what’s happening, and what you need to implement:

* **Instructions:** Instructions are printed to the control pane at the beginning of the program. (You can’t see these in the movie, but you will need to be sure to have them in the program, so that the TF who grades your program knows how to run it.)
* **Image Creation:** Whenever the user types the letter “e” (for elephant), “f” (for frog), or “s” (for seal), a picture of that animal appears at the mouse location and starts moving in a random direction.
  + If the mouse is too close to the edge (within some number of pixels that you can define – I’ve used 30), no image is created.
  + There is a maximum number of images (which I have set to 100 but you can define your own maximum).
  + You can use the images I’ve used, or you can use your own images (at least three different pictures and one explosion image, but it’s fine if you want to have more). My images can be downloaded from:  
    http://www.csee.umbc.edu/courses/undergraduate/CMSC101/  
     Fall2013/Handouts/elephant.jpg  
    http://www.csee.umbc.edu/courses/undergraduate/CMSC101/  
     Fall2013/Handouts/frog.jpg  
    http://www.csee.umbc.edu/courses/undergraduate/CMSC101/  
     Fall2013/Handouts/seal.jpg  
    http://www.csee.umbc.edu/courses/undergraduate/CMSC101/  
     Fall2013/Handouts/explosion.jpg
  + When any other letter is typed, it is ignored.
* **Image Crash:** Whenever two images crash into (touch) each other, they explode! Each image is replaced with a picture of an explosion, which stays in place for a short time (I have it set to 1 second) before disappearing.
  + In the movie, sometimes it looks like an image just exploded by itself. That’s because I moved the mouse on top of that image and typed one of the image characters, which caused a new image to appear and immediately crash (before you can actually see it on the screen).

## A Word on Difficulty and Design

This program is fairly complex. We will be spending one class day (on Thursday, September 26) doing an in-class design and implementation laboratory to work through some of the basic functionality required for this assignment (loading images, placing them on the screen in response to keystroke commands, storing them in an array, and “expiring” them after a period of time). After that class, we will distribute a “shell” program (CrashBasics.pde) that includes those implemented functions as a starting point for the assignment.

As a reminder, the “CrashBasics.pde” program includes:

* Variable (actually constant) definitions for a number of different running parameters – you can change any of these values that you want to, in order to get your program looking the way you like it.
* Definitions of image variables.
* An index variable, nextImage, for how many images have been placed on the screen (and not yet exploded).
* Six different arrays that are used to contain information about the images:
  + images[] – the pictures themselves
  + ypos[] – the y position of each image
  + xpos[] – the x position of each image
  + xdir[] and ydir[] – the x and y directions in which each image is moving
  + times[] – the time at which each image was generated

For example, the third image loaded will be stored in image[2]; its x and y position will be recorded in xpos[2] and ypos[2]; its direction vector will be recorded in xdir[2] and ydir[2]; and the time at which it was created (or exploded) will be recorded in times[2].

## Suggested Implementation and Testing Steps

Although you are strongly encouraged to follow this suggested process, it is not a requirement of the assignment. If you create your program in some different way, but it works completely, you can still get a good grade on the assignment. Note, however, that a percentage of your grade (up to 15%) will be based on having a good design (well written, clearly documented code; modular decomposition; avoidance of excessively long functions or repeated code when a separate function could have been written). You are more likely to end up with a good design if you follow the advice given here than if you strike out on your own.

**(1) READ THE ENTIRE ASSIGNMENT BEFORE YOU START! If you haven’t done this already, *do it now*.**

**(2) Download and Test**

Download and test the provided code (CrashBasics.pde) to make sure that it works properly: you should be able to type an ‘f’ anywhere in the display, and it should draw a frog there, which disappears after one second. *You should* ***only*** *change the parts of the program that are described below – these places are all clearly indicated by comments. It is strongly recommended that you not change any of the other functions or add any new functions! (Of course, if you decide to do extra credit, you may need to make some further changes.)*

**(3) Stop, Think, and Design Before Implementing!!**

Sit down (with your programming partner if you’re working in a pair) and design the top-level functions that are missing. ***You are not ready to open Processing and start typing code until your design is complete. Do not “design by programming” – you should plan the algorithm you are going to use before you write code, and your code should be written on paper before you type anything in!***

* 1. Look at ~~keyPressed() and~~ removeOldExplosions() and make sure you know how to modify these two functions – they are very simple changes that should be straightforward if you understand the code that’s already there. *Note: keyPressed() has been extended to include the “e” and “s” commands for you.*
  2. Design explodeTouchingImages(). This function needs to look for pairs of images that are touching (which you can check by calling intersectingImages()) and replace them with explosions (which you can do by calling explodeImage()).
  3. ~~Modify createImage() to initialize the x and y directions. You should use the random() function with two arguments that represent the range of “speeds.” I used the range -2 to +2 in both the x and the y directions, but you can experiment with other ranges and use whatever you think is most aesthetically pleasing.~~ *Note: This function has now been provided in CrashBasics. You do not need to write it.*
  4. Design moveAndBounce(). This is the longest and trickiest function. It needs to update the xpos[] and ypos[] arrays so that all images have new positions to draw, and update the xdir[] and ydir[] arrays to prevent an image from moving off the edge of the display in any direction.  
      Even after you’ve thought this function through on paper, you’ll still probably have to go through a “test, experiment, debug” cycle once you sit down and start entering your code in Processing in order to get it right. That cycle will be ***much shorter*** if you really think it through at this stage.   
      Here are some questions you’ll want to ask yourself:
     1. How do you detect when an image moves too close to the edge of the display? Is this test different for top, bottom, left, and right edges?
     2. What happens to the x and y directions (velocity) of an image when it moves too close to the edge of the display? Is this different for top, bottom, left, and right edges? (Hint: You don’t need any trigonometry (sines or cosines) in order to change directions correctly, just the negation operator!)
     3. How do the x and y positions change between time steps? What should the new position be at the next time step?
  5. Design explodeImage(), which has to update several arrays to replace the “exploded image” with the “explosion” picture, stop it from moving, and reinitialize the times[] array to start the clock ticking on the explosion timeout.

**(4) Program and Test Incrementally**

Now (and *only* now that you’ve completed your design and thinking process!!) are you ready to start programming and testing your code. You should not just type everything in and then test it all – that strategy will make it *much* harder to figure out what’s going on if anything doesn’t work the first time.

1. ~~Extend the code to load multiple images in setup() . You won’t see any visible changes after you’ve written setup(), but you should test your program anyway, just to make sure you haven’t broken anything!~~  *setup() has already been changed to load multiple images in CrashBasics.*
2. ~~Extend the code to draw the various images you’ve loaded when you press the appropriate key in keyPressed(). Test this functionality before moving on. (Note that the other images won’t disappear, since the current code only looks for frogs to remove. That’s OK – you’re going to change that anyway!)~~ *keyPressed() has already been changed to draw multiple images when you press the appropriate key.*
3. Extend the code in removeOldExplosions() to remove explosions instead of frogs. Test this functionality. Now the frogs don’t go away, either – in fact, nothing goes away. That’s OK – you don’t have any explosions to remove yet!
4. ~~Fix createImage() so that it initializes the movement directions. Test this even though you won’t see any visible change yet.~~  *createImage() has been written and provided for you.*
5. Implement and test moveAndBounce(). Make sure this is working ***completely*** before moving to the next step!
6. Implement explodeTouchingImages(). Since you haven’t yet implemented explodeImage(), nothing will happen when you identify touching images to remove. So you can add some print statements to your code to make sure that it prints something out when two images touch each other in the display. You may not be 100% sure that this is working correctly before moving on to explodeImage(), so be sure you consider whether you’ve done anything wrong here if something’s not working in the next step.
7. Implement explodeImages() and test it to be sure that the images are exploding correctly.
8. ~~Add the instructions to setup() – you should do this last to be sure that the instructions describe how your program actually ended up working.~~ *Instructions have already been added to setup() – you only need to change them if you’re loading different images or implemented some other variant.*
9. Once everything appears to be working, be sure you run your program a bunch more times, experimenting with boundary cases (typing letters when the mouse outside the frame, creating a really large number of images, typing letters that aren’t supposed to do anything).

Congratulations! At this point, you should have a working program! Be sure to ***add your name(s)*** to the top of the program, and ***clearly comment*** your new code, and any changes you may have made to the provided code.

**(5) Submit Your Assignment**

1. Submit the .pde program file on Blackboard as an attachment, in the “PA3” assignment. If you worked in a pair, *one*of you should submit the .pde file, and *both of you* should submit a comment that you worked in a pair, and naming your partner (by name and UMBC email so that we can match you up easily). If you used any different images, be sure to upload those image files as well by using a .zip archive.

* *When you save your Processing sketch, name it <username>\_crash (Example: mariedj\_crash).*
* *Upload <username>\_crash.pde into the assignment “PA3.”*
  + *IMPORTANT NOTE: If you have any companion files (e.g., images other than the standard provided ones), you must instead submit an archived sketch that contains your entire sketch directory with the images and .pde file. You can create this zip file by using the “Archive Sketch” option under the “Tools” menu.*

1. *Turn in (in class on the due date) a* ***hard copy*** *description of your project. It should have the following elements:*

* *Your name, user name, and UMBC email address.*
* *A very short summary of your program’s capabilities, including any extra credit that you implemented.*
* *The names of anyone you discussed the project with. If you discussed it with no one, say so.*
* *A signed statement that you wrote your own code.*
* *A printout of your <username>\_garden.pde program.*

As in the first Processing assignment, if your program doesn’t work completely or correctly (or at all...), you should prepare a document (.docx in Word or .txt in NotePad) that explains clearly and thoughtfully what you *have* accomplished, what isn’t working, what you’ve tried to fix it, and any thoughts you have about where it went wrong. Although you cannot receive full credit for this assignment unless your program is working correctly, a good explanation of where you’re stuck will reduce the penalty for incorrect programs. Upload this explanatory document into Blackboard as an attachment. If you worked in a pair, you should each write and submit *your own document* – you can discuss it together, but the actual words you write should be ***your own***. You should still submit the .pde program (one of you) and a note about who your partner is (both of you).