In A4, you will learn how to handle 2-dimensional graphical objects (also called shapes) and design efficient algorithms for two tasks.

1. Checking whether two 2D shapes overlap (collision detection)
2. Checking whether a point of the plane belongs to a shape.

Keep track of the time you spent on this assignment. We will ask for it when it is time to submit.

1. Grading

Solutions will be graded on correctness, the quality of the algorithms, and style. A correct program compiles without errors or warnings, and behaves according the the requirements given here and in the comments of the code. A program with good style is clear, concise, and easy to read. You are expected to follow the style guidelines given for this course on the course website.

2. Collaboration policy and academic integrity

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and at least by the day before you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something before the group is formed: one proposes, the other accepts.

If you do this assignment with another person, you must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns "driving" —using the keyboard and mouse.

With the exception of your CMS-registered partner, you may not look at anyone else's code, in any form, or show your code to anyone else, in any form. You may not show or give your code to another person in the class. While you can talk to others, your discussions should not include writing code and copying it down.

3. Getting help

If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders.

4. Installing the code

The release contains source code (files with extension .java) in the A4/src directory, image files (with extension .bmp), and a jar library (vecmath-1.3.1.jar). Follow these steps to install the code:

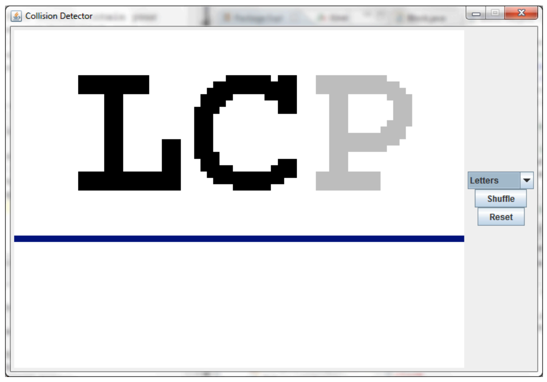
1. Unzip file A4release.zip into your Eclipse directory A4. Note: All .java files go in the default directory.
2. Create an Eclipse project named *A4*. Right click on the project and select "Refresh" from the menu.
3. If the project has errors, add the jar library to the build path: Inside Eclipse, right-click the project, go to "Properties", then to the tab "Libraries", and then press the "Add External JARs" button.

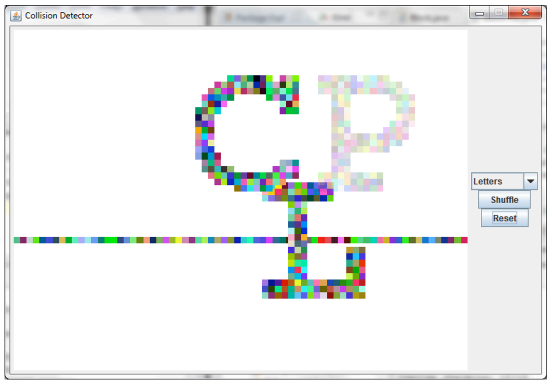
The main method for the program is in file GUI.java. The first lines of this file contain two static fields: WIDTH and HEIGHT. You can change the values there to make the window fit your screen well.

5. Shapes

The main functionality of the program is the following:

- First, it reads images from bmp files. Such images are simply 2-dimensional arrays of pixels, where a pixel is described by the three numbers: the red, green, and blue color component
- Next, the image is processed by the "image blocker", the code of which is found in file ImageBlocker.java. The image blocker finds the objects (shapes) in the image and returns them as a collection.
- A user interface is provided that allows you to click on shapes and move them around. We have already provided naive implementations for detecting overlap of shapes and checking whether a point lies in a shape.
  - When you click on a shape it should become semi-transparent. Below, in the diagram to the left, P has been clicked on. If you click on it again it gets restored to its original color. There are a few lines of code missing for this feature, which you have to add.
  - When shapes overlap, they are highlighted. In the figure to the right below, the shapes overlap. This feature is already fully implemented in the release code.





Consider the example of the image containing "LCP" shown above. This is a small image of width 77 pixels and height 33 pixels. The image blocker will identify 4 objects in it: the letter L, the letter C, the letter P, and the blue line at the bottom.

From each pixel the image blocker creates a "block", which you should think of as a little square filled with a specified color. A shape is a collection of blocks. The naive algorithms we have provided may be so slow that you will want to work only with image "Letters" while developing your algorithms. The program may be unresponsive if you try to work with the larger images initially. Only after you have successfully implemented the faster algorithms will you be able to use the larger images. We have provided various images of increasing complexity:

Letters < More letters < Mario small < Very angry birds < Mario

6. Coordinate Systems

Three different coordinate systems are relevant to this program.

1. Coordinate system of an image file: Let W be the width of the image (number of columns) and H be the height of the image (number of rows). The pixel with coordinates (0,0) is the top left pixel. The pixel with coordinates (W-1, H-1) is the bottom right pixel. The x axis is horizontal (direction left to right) and the y axis is vertical (direction top to bottom).

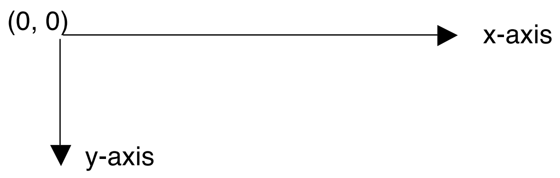
2. Coordinate system of a "blocked image": This is the coordinate system in which the objects created by the image blocker exist. Suppose that in the bmp file $W \geq H$ (analogous things hold in the case $H > W$). The integer interval 0..W-1 is scaled down to a real interval [0; 1]. The origin is the point with coordinates (0, 0) and the direction of the axes is as in (1). Each pixel of the image has been converted into a block of dimensions (1/W) x (1/W).

The quantity called *halfwidth* is defined to be equal to *1/2W* (half the size of the width/height of a block).

For example, the pixel with coordinates (0, 0) in the bmp file is converted into a block (a little square) with center (1/2W) x (1/2W) and dimensions (1/W) x (1/W). Equivalently, its top-left corner is the point (0, 0) and its bottom-right corner is the point (1/W, 1/W).

3. Coordinate system of the window of the GUI (we call this the "canvas" in the code): This is a scaled version of the coordinate system described in (2) to make it fit well into the window.

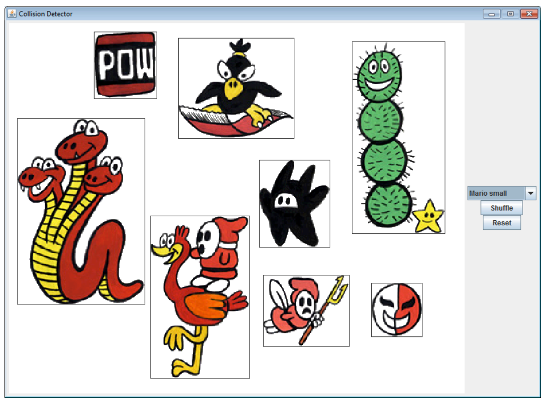
Important: Always keep in mind that origin of the coordinates systems is the top-left corner. The direction of the x axis is from left to right, and the direction of the y axis is from top to bottom. as shown below.



7. Project Structure

A significant part of your work for this assignment will be to read the release code and figure out how it works. You may need to find the relevant Java documentation on the Web and read it. Successful completion of the assignment requires you to understand well how the existing code works. Here is a brief description of classes:

BoundingBox: This class models a rectangle. It is used to describe "bounding boxes" or "bounding rectangles" of shapes, i.e. rectangles that enclose shapes. See the figure below.



Shapes drawn with their bounding boxes

Block: The basic unit of a shape. Intuitively, it represents a pixel of the original image in the bmp file.

ButtonPanel: This class creates the buttons you see on the window.

Canvas: This handles drawing in the "canvas", the white area of the window.

ImageBlocker: Identifies shapes in an image. You do not need to understand how this class works to complete the assignment successfully.

Vector2D: Describes a 2D vector (or a 2D point) with components of double precision.

Shape: Describes a shape as a collection of blocks.

GUI: Contains the code that starts up the program.

BlockTree: Contains a hierarchical tree structure used to speed up the main operations of the program (overlap detection and containment queries).

BoundingBox, BlockTree, and Shape are the main classes you need to modify, in that order. You will also have to do some slight modifications to class Canvas. Do not change anything else in the code. *Every point inside the code where you have to change/add something has been clearly marked for you, using a comment that starts with "// TODO".* You can see them marked in the lefthand column

8. Class BoundingBox

The methods that have to been implemented have been clearly marked in the release code.

Math background: A 2D vector is a pair of two numbers v = (x, y). We add vectors by adding their components:

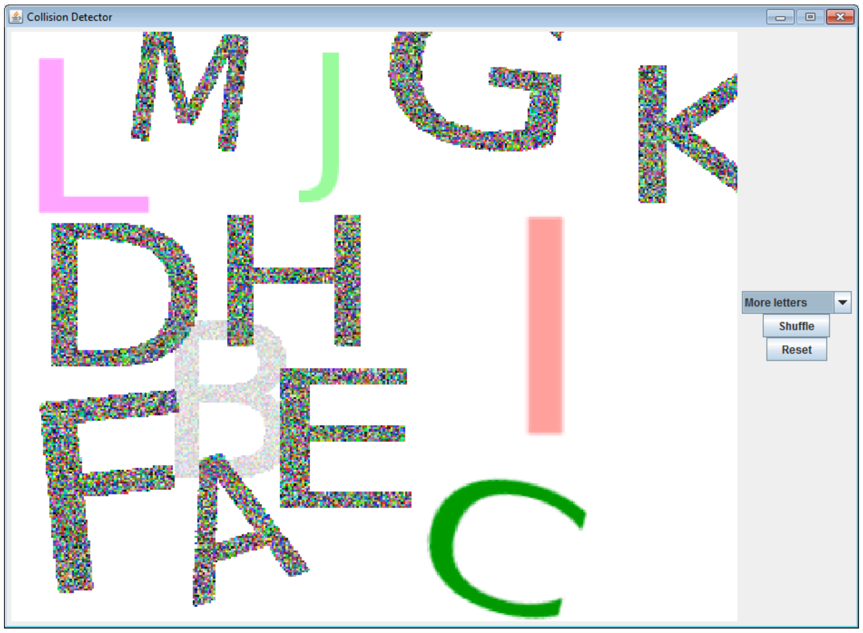
$$(x, y) + (x_0, y_0) = (x + x_0, y + y_0) .$$

A 2D point is also a pair of two numbers p = (x, y). We can identify a point with the vector that starts at the origin (0, 0) of the coordinate system and ends at the point. When we say a point p = (x, y) displaced by a vector v = (d_1, d_2) we mean the point ($x + d_1$, $y + d_2$).

9. Class Shape

The methods that have to be implemented have been clearly marked in the release code, and all important fields are already defined. Note that the object maintains a displacement (as a Vector2D) that describes the translation of it (and its BlockTree) relative to its original screen position.

Remark: This is a very "light" class with little code. The actual work of implementing the good algorithms for collision detection and containment queries will be done in the class BlockTree, not here.



Screen shot of a fully functional program

10. Class BlockTree

The methods that have to be implemented have been clearly marked in the release code. In this class you will write the code to implement fast collision detection and containment queries. The main idea is to construct a tree data-structure to speed up the search. This tree can be thought of as a binary search tree for collections of blocks. Every shape holds such a tree structure in order to organize its blocks.

• In order to built the tree: The blocks of a shape are partitioned into two parts, and for each part a tree is built recursively. If there is only one block, the tree is a single node containing this block.

• Hint: You need to figure out a good way to split a collection of blocks into two parts. We suggest you start by calculating the bounding box of all the blocks. Then, "split" the box in half along the larger dimension, and put the blocks into these two areas.

• After the tree has been built: You can speed up the search for overlap testing and containment queries significantly by making good use of the bounding rectangles of (sub)collections of blocks. Thus, if two shapes have non-overlapping bounding rectangles, then the shapes do not overlap.

11. Class Canvas

There are very few things to be added/changed in this class, and we have marked them all in the release code.

• *Shuffling*: This can be implemented correctly only after you have implemented your tree structure. All the code has already been written for you and it will start working on its own after you implement the tree.

• *Clicking on shapes*: When you click on a shape, it should become semi-transparent. When you click on it again then it should be restored to its original color. You need to add a few lines of code in method mouseClicked to make this happen.

• *Fit images well in canvas*: Method setNewImage contains some code to un-comment in order to make the images fit better in the canvas. All the code has been already written for you. You need to implement the tree structure first.

12. Goal

In a successful implementation of the algorithms, the program will be responsive even with the big "Mario" image. This means that you will be able to drag around shapes and see colliding objects being highlighted with little observable time delay, as in the figure shown on the previous page.

Note: Depending on your machine and implementation, you will likely find that, when you complete the assignment, your overlap tests will be far faster than the graphics display. This is fine. The Java2D graphics code is not highly optimized for drawing many small pixels as individual rectangles, but it is sufficient for this assignment.

11. Submission

First, fill in information in the comment at the top of class GUI.java. The hours hh and minutes mm that you spent doing this assignment, your name, and your netid. If you want, tell us what you thought about this assigment. Was it interesting playing with GUIs? Did you see how data structures can be use dot speed up programs, in fact, make them feasible? Was it too hard? Too easy? What could be improved.

If you discussed code extensively with people, tell us who they are.

Finally, tell us about issues you have with your program, anything you think needs clarification.

Second, put the entire src subdirectory into a zip file. It should contains all the .java files needed for your program. Do not include the bin subdirectory, containing files ending in .class, the deps subdirectory, containing the .jar file, or the images subdirectory, containing files ending in .bmp.

All .java files should compile and conform to the prototypes we gave you. We write our own classes that use your classes' public methods to test your code. Even if you do not use a method we require, you should still implement it for our use.