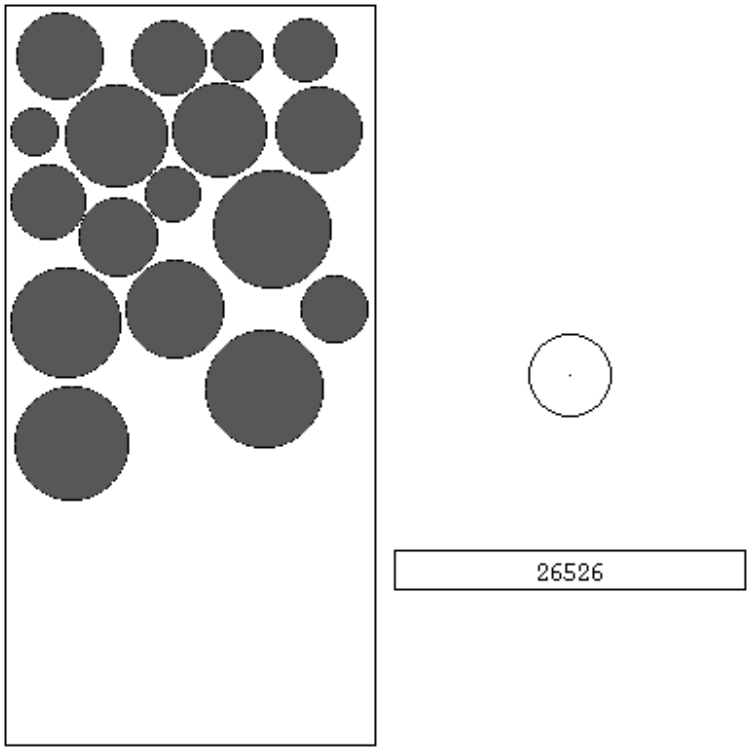
Laboratory 8: Packing Bins

Objective. To implement a game with unbounded memory, using arrays or vectors.

Discussion. We frequently write programs that have to remember an increasing amount of data. In this lab, we will write a program that challenges you to stuff as many disks into a rectangular box as possible. The program appears as follows while running:



On the right side of the screen, disks of random diameter appear, one at a time. When pressed, the mouse is able to drag the disk into the rectangular bin, on the left. Releasing the button drops the disk at that location. If the disk overlaps an existing disk, it returns to the start location. If, when dropped, the disk laps outside the bin, the game ends. Otherwise, the disk is permanently placed in the indicated location. Each placed disk increases the score by the approximate number of pixels covered.

The program must, of course, remember where all the previously placed disks are located. For this reason, you will need either an array or a `Vector` to store the disks.

Once constructed, the program has the interesting feature that there is no obvious winning strategy to placing the disks in a manner that maximizes the number of pixels covered.

Procedure.

Before you approach this program, you should consider whether you wish to use arrays or `Vector`s to store the disks. If you use an array, you should be careful to select a bound on the number of disks that is not so large as to be wasteful of memory, and not so small as to potentially cause problems if large numbers of small disks are to be placed. You should then follow these steps:

1. On graph paper, layout the drawing window. This will help you in determining the size of the score box, and the potential size of the disks you are to place.
2. Write (or reuse) a function that computes the distance between two points. Why is this necessary?
3. Write a program that draws the screen and score box (with a score of zero).
4. Extend this program to pick a random circle place it within the bin. The program should stop after this one placement.
5. Extend this program to include the store for previously placed disks. Add the disks to the array, but don't worry about checking for overlap. Stop when a disk falls outside the bin.
6. Finally, add the check for overlapping disks.

Thought questions. Consider the following questions as you complete the lab:

1. Suppose you were placing `Rect` objects. How would you check for overlap of two `Rect`s?
2. Suppose you wanted to help the user place the disks correctly. How would you draw all the target pixels where the center of the disk could be located within the bin?
3. It is possible to construct a `Random` with a starting random number, called a *seed*. The seed determines the enire sequence of random values selected from that point on. Why might we want to have the user specify a starting seed when the program is run?
4. (Hard.) What percentage of the area would you expect to be able to cover with disks?