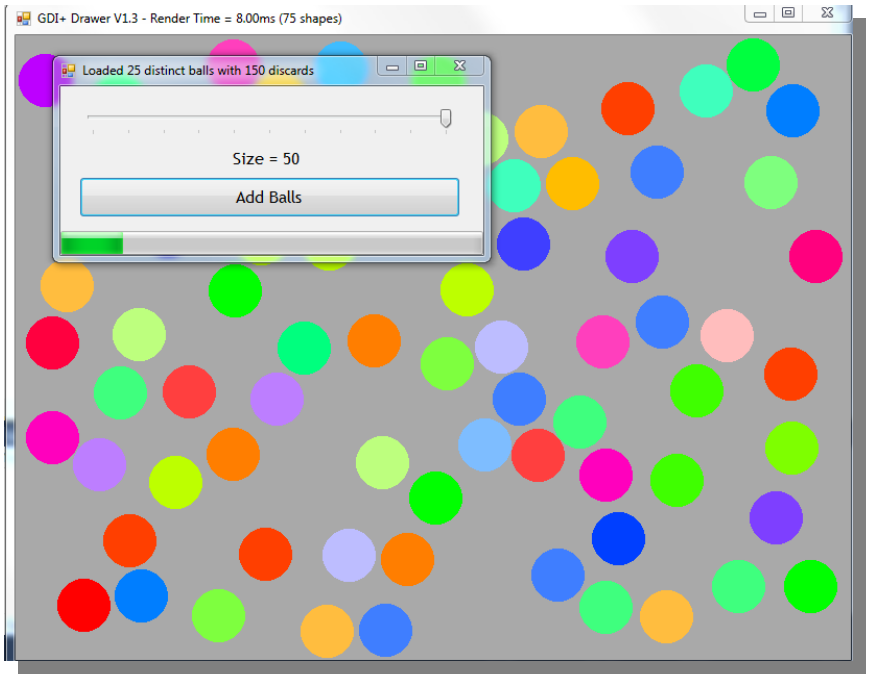
CMPE2300 – ICA04 - Don't Be Touching My Balls

In this ICA you implement a simple class implementing an Equals() to ensure your balls don't overlap.

Create a new Windows form project for this ica, fashioned very similar to ica03. You must start this ica from scratch, with an appropriate name. Construct a UI for your form fashioned after the following capture:



Add a new **class** Ball which will contain the following instance and static fields, same as ica03 as indicated with [3]:

- a static field of type CDrawer, use an initializer
- a static field of type Random, use an initializer
- an instance field of type int, representing the Radius of the ball - provide a public property as well, set only, take the absolute value of the value and set the Radius
- an instance field of type Color for the ball color
- an instance field of type Point for the ball center

Add a **static constructor** to your class, where you finalize CDrawer properties, setting the background color to a [RandColor] random color and continuous update to false.

Add a **instance constructor** accepting an int representing the ball radius

a. Initialize the color to a [RandColor] random color
b. Initialize the radius to the supplied argument using the property (to protect the value)
c. initialize the location to an X,Y value to random values ensure no boundary overlap, **You must use the CDrawer object scaled width/height here

Add a public instance method called **AddBall()**, returns nothing, accepts nothing. This only adds a CenteredEllipse to the drawer using the instance location and color, and radius.

Add a public static property - a virtual one, not tied to an actual field at all, called **Loading,** supply set only, no get. As before, this property acts like a "switch". This will be implemented by checking the Load "value". If the user set it to true, clear the drawer, if he set it to false, render the drawer ** This will be contingent on the continuous update of the drawer being false

Add a private static helper function, it should accept 2 Balls as parameters, and return a double. It will simply return the absolute difference between the center of the 2 supplied Ball arguments. It will be used in Equals().

Add an **override** for **Equals**, return true if the supplied Ball would “touch” or overlap the invoking ball. Using your helper, determine if the absolute distance is less than the combined radii, if so, an overlap exists, and we shall “consider” these to be equal (true).

We will use the List.Contains() method, which normally would be using identity semantics when looking for a match, but our override will force value semantics allowing Contains() to be true for any other Ball that triggers true via an Equals() invocation.

Main Form :
add the trackbar (minimum = -50, max = 50), and button as shown

add a handler for the track bar for scroll, the current value of the trackbar will be used to set the Radius of the Balls - use the property here to ensure input filtering of input

add a handler for the button, this will add 25 “untouching” Balls to our list and show them

a. While(Hint, hint), you have added less than 25 balls and you have not discarded 1000 touching balls, create a ball, then use list.Contains to determine if the new ball would be touching any existing balls, and increment appropriately.
b. The Loading property would be used here, and upon completion, update the Title Text as indicated.
c. The progress bar shown has a range of 0-1000, and reflects the discarded balls as the list is populated. Note that you can discard 1000 per button press not 1000 total.
d. Upon completion of the while, Render() all the balls added during your loop.