README for Homework 3

Rong Kang Chew (rchew)

What solver you are using, including references. If you have changed the way the solver works from the documented algorithm, please describe the differences.

Hudson's Lazy Constraint Solver is used to solve constraints.
The system allows for primitive values or one-way, singular constraints per defined user-settable attribute for each GraphicalObject.

When declaring a constraint, the attribute being constrained must also declare what other attributes from which objects is it depending on.

The system also uses a "changed" marker to note when its dependencies has changed in value. To allow for propagation of changes to be noted, the developer needs to additionally provide references to the object of the attribute being constrained, and the name of the attribute.

Evaluation of the constraint function is only done when the value is requested (e.g. usually by a draw, bounding box, or another constraint using that value). Propagation, if necessary, happens after a setter is used or evaluation is done.

How you needed to change the design from HW2 to fit in your solver. For example, did you change the high-level definitions of graphical objects or their attributes? Also include notes on how you fit the solver into your implementation, so I can find it in your code files.

- Getters and Setters had to be established for all user-settable attributes in order to "hide" the evaluation primitive/Constraint when Getting and propagation when Setting system.
- The use of the .valueOf() function in JavaScript was helpful. The function just returns the same value for primitives, while a method with the same signature that performs the evaluation was added to the Constraint object so that the Getter could just call _x.valueOf()
- GraphicalObjects needed to maintain a list of its dependents so that changes could be propagated up and also allow for cyclic dependency checking.
- Cycle checking is done before any propagation or evaluation. Any cycles detected will result in an exception being thrown.

- I had to move some code that evaluated the positions with groups in mind into the Getters for x and y.
- I decided not to allow async for the Constraint function for simplicity. Some methods were changed back to non-async as they could not be used in the function (getBoundingBox, moveTo). These functions were originally async to accommodate issues with image loading.
- The current solver cannot protect other objects from side effects as the function is entirely up to the user. This could have unexpected effects.

A programming guide that documents for a user of your API how to write a constraint. For example, if I want to write a custom constraint, what do I do?

User-settable attributes (e.g. x, y, color, lineThickness etc.), can either have a primitive value or a Constraint object.

The constructor for a Constraint object as follows:

```
Constraint(originObject, originPropertyName, initialValue = 0, func,
dependencyOnList = [])
```

where

- originObject and originPropertyName are the object and property being constrained
- initialValue is the value of the property to use before first evaluation
- func is the user-defined, non-async function returning a value (of the expected primitive type for the property) to evaluate for the property
- depedencyOnList is an array of DependencyOn(object, propertyName) objects that describes the other objects and their properties that are depended upon for this Constraint

For example, if rectangleB is to be constrained to the immediate right of rectangleA:

```
rectangleB.x = new Constraint(rectangleB, "x", 0, function() {
             let bb = rectangleA.getBoundingBox();
             return bb.x + bb.width;
         }, [
             new DependencyOn(rectangleA, "x"),
             new DependencyOn(rectangleA, "width")
         ]);
```

Evaluations are done in a lazy manner, i.e. the system will only re-evaluate the function if it detects a dependent property has changed.

Do not set dependencies that will result in a cycle (and therefore a stack overflow), the system will throw an exception if detected.

Notes:

- Using moveTo for line will override x1,y1 and x2,y2 with fixed values from moveTo(x,y). X or Y cannot accept dependencies.