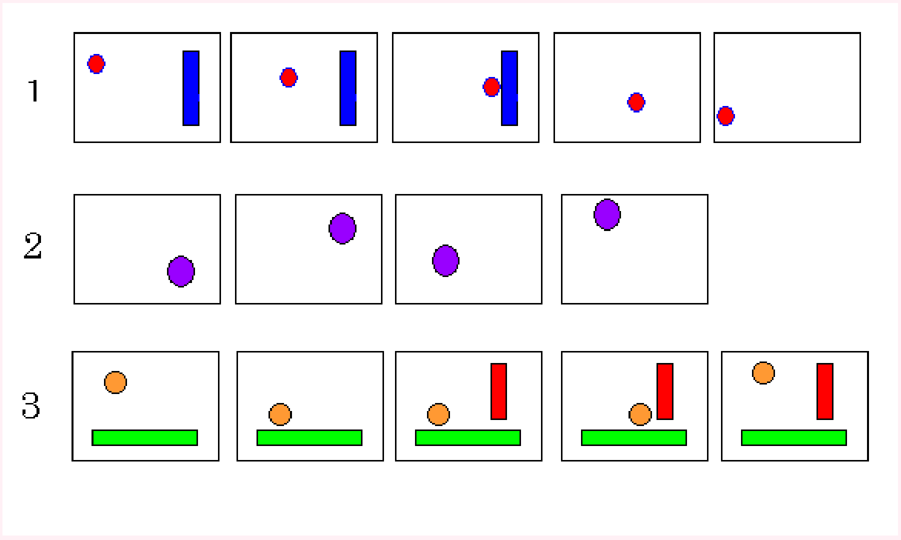
CS 1102 (A14) Individual Project
Implementing an Animation Language

Phase 1 (design) due Monday, September 29 (11:59pm)
Phase 2 (implementation) due Sunday, October 12 (11:59pm)

Project Description

You must design and implement a language for animating graphic objects on a canvas. Graphic objects can move smoothly or jump around the canvas, optionally stopping when a graphic object bumps into something (like an edge of the canvas or another graphic object). Graphics can also be added or removed during an animation. Here are three examples of animations.



- The first sample shows a red ball moving at a angle towards the wall until it hits the wall. At that point, the wall disappears and the ball moves back towards the left edge of the canvas, stopping when it hits the left edge of the canvas.
- The second sample shows a purple circle jumping to random locations around the canvas until it hits the top edge of the canvas.
- The third sample shows an orange circle dropping straight down until it hits the green rectangle. At that point, the red rectangle appears and the circle moves right until it hits the red rectangle, after which the orange circle jumps to a random location.

For this project, you will develop:

1. A language for specifying animations such as these and
2. A program (interpreter) that will run an animation written in your language, displaying it on the screen.

You will do the project in two stages: a language design stage, followed by an implementation stage. Each stage has a separate due date.

Project Goal

The goal of the project is to make sure each student can define, design, and implement a domain-specific programming language. **This is an individual project -- you may not work with your homework partner, or any other students, on this project** (see the collaboration policy for more details).

The Design Phase: DUE Monday, September 29, 11:59pm

For this phase, propose data definitions for a language for animations. At a minimum, your animation language must meet the following requirements:

- Graphics must include at least circles and rectangles. You many optionally include other images (such as gifs).
- Individual commands must include moving, jumping, and deleting graphics that are already on the canvas, as well as adding new graphics to the canvas. These commands must be able to repeat until a graphic hits either an edge of the canvas or another (specific) graphic.
- The programmer should be able to specify the rate at which an object moves along each of the x and y axes, as well as each

object's initial location on the canvas. Here, "rate" means how many pixels the object moves between each frame of the animation.
- Final project grades of B of better will need to support repetitions of entire blocks of commands (see Grading section below for other criteria for different grade levels).

Submit both the proposed language and examples showing how to represent all three sample animations, plus an animation of your own choosing, in your language.

You do not need to be able to run animation programs at the end of this stage. A nice syntax (ie, macros) for the language are not needed at this stage either (you can add those in the implementation phase). All you need to submit are the data definitions and examples of data that you need to capture animation programs, as you did for Homework 4.

Do not submit any function definitions for this phase!

HINT: Think of the slide show language we developed in class (including the overlay extension in lab) as a very slow and jerky animation. This should give you a good idea of how to think about designing this language.

What to Turn in

Submit an electronic file *design.rkt* (under turnin name Project Design) containing your work for this phase.

The Implementation Phase: DUE Sunday, October 12, 11:59pm

For this phase, you must provide a function `run-animation` that takes an animation in your language and runs it (displays the animation on the screen). Your animation should happen in one window over time---you are not trying to produce a sequence of still frames as shown in the samples.

You may decide to change or enhance your original language design as you write your interpreter. That's fine (even expected---implementing helps you assess your design decisions). Your project report (see below) should describe and explain the reason for all changes you decided to make from your original design.

N.B. In this project, you will use a modified version of the 'world' graphics teachpack (called world-cs1102.rkt). Unlike the graphics teachpack you used for Homework 2 (the fire plane assignment), which automatically displayed successive frames for you, this modified teachpack gives your language interpreter direct control of frame updating. Here are the three key functions in the modified teachpack that you need to use:

- `(big-bang width height rate init-world)` initializes a drawing canvas with the given width and height. The `rate` and `init-world` arguments won't be used by your implementation (but must be supplied). Use 1/28 for `rate` and a dummy value (e.g., `true`) for `init-world`.

- `(update-frame image)` replaces the contents of the canvas with the given image. An image is the same as a scene, so you can use the `empty-scene` and `place-image` functions that you used in the Homework 2 to build your frames as before.

- `(sleep/yield seconds)` delays the drawing of the next frame by the given number of seconds. This can be used to control the rate of your animation. Values around 0.25 work well for this in practice.

To use world-cs1102.rkt in your project, download it here and save it in the same directory as your project file. Then:

- If you are using the Advanced Student language, in the DrRacket menu, select Language > Add Teachpack > Add Teachpack to List > navigate to world-cs1102.rkt > Open > Ok.

- If you are using the Pretty Big language (necessary for defining macros), then include the following at the top of your project file:

```
(require "world-cs1102.rkt")
```

The Project Report

Provide a text file with answers to the following questions:

1. What must the TA do to run your program? Provide concrete instructions (such as "execute (run-animation animation1)"), including a list of the animations you defined as your test cases. The staff won't grade a program that they can't run.

2. What is the status of your implementation? Explain which features/aspects work and which don't. If you didn't get the repeating commands to work, for example, say so. This gives the staff guidelines on how to test your system.

3. How have you changed your design since the version you submitted for the design deadline? Explain the changes and why you made them (i.e., I found I couldn't do X because of problem Y with my earlier definition). We're interested in seeing what doing the implementation taught you about the language design.
4. What, if anything, do you think could be cleaner in your design or implementation? If you are satisfied with your design, say so. If you think certain aspects should really be easier to use, easier to write, etc, explain those aspects and what you'd like to see different. No danger of losing points for honesty here (you'll only lose points for problems that we can detect without reading your report) -- we just want to hear your assessment as we determine our own.

What to Turn in

1. A file *project.rkt* containing your work for this phase.
2. A file *report.txt* containing your project report. Please submit these in plain text, rather than in Word or PDF format.

Submit these via turnin, under the name Project Final

Grading

In general, the design phase counts for 25% of your project grade, the implementation phase (including final language design) for 70%, and your project report for 5%.

Design Phase Grading

We will grade your language designs on a 4-grade scale (check+, check, check-, no credit). At this stage, we're looking to see whether you thought out the design phase well -- did you identify appropriate data and commands? Does your design adequately support the given examples? Does your work demonstrate that you know what data definitions for languages look like?

There's no single right answer for this part, and while we will make suggestions on your designs, we won't give you a single right answer to follow when doing your implementation. Part of the exercise is for you to have to work with, and perhaps revise, your initial language design when it comes time to implement your animation system. Grades in this phase are more about how well you cover the sample animations than your low-level design decisions (which we fully expect to change as you start to implement the project).

Implementation Phase Grading

In the implementation phase, we will be looking at your final language design and its implementation. More specifically:

- A grade of C (passing) on the project requires a reasonable language design and a working `run-animation` function that support the minimum language requirements described in the design phase description. All three sample animations should execute properly in your language.
- A grade of B requires that you satisfy the requirements for a C and support repeating sequences of commands (rather than just individual commands). Include an example animation program that uses repeated sequences of commands.
- A grade of A requires that you satisfy the requirements for a B grade, have a very good language design, **and** add a clean, macro-based interface to your language (as we illustrated in class with the macro-based design for the slide show language).

Some General Notes on Grading

- You are welcome to add extra kinds of animation to your language, but new features will not make up for missing required elements of your language. **Adding features while your program isn't yet robust will cost you points**.
- We will grade your report for writing (complete sentences, spelling, punctuation, clarity, etc) as well as technical content.
- We will look for whether you followed good coding practices, such as uses of helper functions, in your prototypes.
- We will **NOT** deduct points for details regarding the relative coordinates of graphics, such as whether a circle hitting a rectangle overlaps the rectangle slightly before your program detects the hit or whether a bounce happens a couple of pixels early. We care that you see how to structure a language implementation, not that you precisely implement graphics-manipulation algorithms.
- We expect you to follow the Homework Expectations on this assignment, with two exceptions: (1) you may use `begin` where appropriate, and (2) you may use set! to store the time at which an overall animation started or to maintain a global list of variables in your program (but not for anything else). Contracts are expected. You do not need to include the test cases for the

individual functions (but we hope you are testing as you go along nonetheless).

Collaboration Policy

This is an individual project. Collaboration is not permitted on this assignment (not even with your homework partner). The course staff are the only people you may approach for help with this project (but do come to us if you need help). You may not ask anyone outside of the course staff questions on any aspect of this project. This includes:

- Asking friends (in or out of the course) for advice on your design.
- Asking friends for debugging assistance.
- Sharing design ideas with other students.
- Looking for ideas on the web.

Violations of this policy follow the general course collaboration policy, and will likely result in an NR for the course.

Why this policy? Given the fairly open collaboration policy on homeworks, this assignment helps us assess how much each student understands of the course material. Since some students struggle in timed situations such as exams, the project gives you a more open-ended setting in which to demonstrate what you've understood of the course material.

This page maintained by Charles Rich based on material originally provided by Kathi Fisler