Build Your Own Animation

Students create a game of their own design using what they have learned so far.

Product Outcomes	• Students make a game or animation of their own design
Materials	• PDF of all Handouts and Page • Slides are not yet available for this lesson • Printable Lesson Plan (a PDF of this web page)
Prerequisites	• Simple Data Types • Contracts • Simple Inequalities • Compound Inequalities: Solutions & Non-Solutions • Piecewise Functions and Conditionals • Introduction to Data Structures • Structures, Reactors, and Animations • Key Events

Glossary

instance :: a specific example of a data structure, with specific values for each field (e.g. - (4,5) is an instance of an (x,y) coordinate

Build Your Own Animation *55 minutes*

Overview

Students apply the Animation Design worksheet to their own, creative animations.

Launch

You've now learned the core tasks that go into building an animation:

- Draw some sketches to illustrate your animation
- Analyze the game elements to identify the information that changes across frames
- Create a data structure to capture those elements
- Write `draw-state` and one or both of `next-state-tick` and `next-state-key`
- Create a reactor to pull it all together

In this lesson, we show you how to use our *Animation Design Worksheet* to keep track of these steps as you build your own animation.

Brainstorm an animation or game that you would like to build. Don't make it too complicated. Start with no more than 4 pieces of changing information.

As we saw in the previous unit, we can always go back and add more elements or details to an existing animation. So keep it simple to get a basic game running, then you can add more later.

Turn to Animation Data Worksheet in your workbook.

Fill in three sketches from your animation. Discuss with your partner whether the sketches you chose have highlighted interesting aspects of your game.

The tables below the animation sketches ask you to identify the elements in your game.

Fill in the first table, noting the elements that are changing and how they are changing. If you have more things changing than there are rows, consider making a simpler animation first then extending (at the end of the lesson).

Have your partner or a classmate check your work — make sure you both agree that you've identified everything that is changing.

Fill in the second table, figuring out data types that capture each piece of information that is changing in your animation. Talk with someone to check your work (and help check the work of others).

The table at the bottom of the worksheet asks you to make a to-do list of which functions and components you will need to write to build your animation.

Mark off which components and functions you expect to need. Think about whether your animation updates on ticks, key presses, or both.

Students should have ended up checking "sample instances", draw-state, and "reactor", plus one or both of next-state-tick and next-state-key. Sample instances get created anytime you have to create a data structure, and every animation or game has an underlying data structure.

Go to the top of the ../../lessons/re-build-your-own-animation/pages/def-data-structure.html.

Investigate

Define a data structure for your game state, with one field for each piece of changing information that you identified in the table of the middle of page 1 of the worksheet. The name of your data structure is up to you, but should reflect the theme of your game (like `RocketState`, `SoccerState`, `OceanState`, etc)

Is your data structure defined?

Write down the sample ***instances*** of your data structure for each sketch that you drew at the top of the first page of the worksheet.

At this point, you could open a new Pyret file and type in your data structure and your sample instances. This would help you check whether your instances and data block are consistent with each other. If you don't have access to the computer right now, you can come back and do this step later.

Sanity-checking each bit of code and examples as you go helps students catch errors early. So typing their work so far in now makes sense, if your class set up allows it.

Now you have to develop whichever functions you marked off on the todo-list on the bottom of page 1 of the worksheet.

Pick one of the functions you need to develop. Follow the design recipe, including working out examples, as you develop each function. Finish and test each function before moving onto the next one.

There are extra design-recipe worksheets in the back of the workbook if you need them to help you remember the steps (domain and range, examples, function, and typing and testing your function).

You need to decide how much scaffolding and help your students need at this point. You can feel free to let them work on their own, or you can encourage them to work through design-recipe worksheets if they still need the structure that those provide. The main goal is to have students tackle only one function at a time, and to make sure it is working before they go on to the other functions.

Finally, we can build and run the animation by defining a reactor.

Add a reactor to your file, then interact with it to run your animation!

Remember that a reactor looks like:

```
??? = reactor:
  init: ???,
  on-key: next-state-key,
  on-tick: next-state-tick,
  to-draw: draw-state
end
```

where you replace the `???` with names and instances that correspond to your game.

Closing

Congratulations! You have created your own animation from scratch. If there are features you want to add, use the extra Animation Extension Worksheets from the back of the workbook to help plan and manage your changes. If you build up an animation one piece at a time, you can get to a fairly complex game in a manageable way.