Unit 7
Complex update-world

Unit Overview

Students continue to combine their use of Cond and Data Structures, this time identifying ways in which the World structure might change without any user input.

Learning Objectives:

- Add detail to their undertsanding of the update-world function
- Identify possible sub-domains which require different behavior of the function

Agenda

5 min	Introduction
20 min	Protecting the Boundaries
45 min	Tests and Results
15 min	Branches in update-world
5 min	Closing

Product Outcomes:

- Students will use Cond in their update-world functions
- Students will identify circumstances in which the functions should behave differently
- Students will define these circumstances - and the desired behavior - in code, as different Cond branches

State Standards See our Common Core Standards Table provided as part of the Bootstrap curriculum.
Length: 90 minutes

Materials and Equipment:

- *Computers w/DrRacket or WeScheme*
- *Student workbooks*
- *Design Recipe Sign*
- *Language Table*
- *The Ninja World 5 file [NW5.rkt from source-files.zip | WeScheme] preloaded on students' machines*

Preparation:

- *Language Table Posted*
- *Seating arrangements: ideally clusters of desks/tables*
- *Write the Ninja World version of update-world towards the bottom of the board, with room to transform it into a cond branch under the function header.*

Introduction (Time 5 minutes)

- *Students should have Ninja World (with dogX, rubyX and catY defined) open on the computer.*
- Let's return to our Ninja World...it's slowly turning into a finished game!
- Look at the code for `update-world`. Raise your hand if you can tell me - in English - what this function does.
  Take a volunteer or two.
- What is `dogX` when the dog is in the center of the screen? 320. According to the code, what will the next `dogX` be?
- What is `dogX` when the dog is on the right-hand edge? 640. What will the next `dogX` be? And the next? And the next? It disappears and never comes back!

Protecting the Boundaries (Time 20 minutes)

- Just like in Bootstrap 1, we need to write a function that checks whether the dog has gone off the right side of the screen.
- Turn to Page 33. Let's work through the first function together. Write a function called `off-right?`, which returns true if a coordinate is greater than...what number? 640.
- We're going to go through the design recipe, just like always: what's the first step? Contract and purpose statement. What's the name of the function? Domain? Range? Purpose statement?
- Alright, let's pick a few example coordinates for our examples: What is an x coordinate that would put our dog on the center of the screen? How do we check whether it's off the right hand side? We said that anything greater than 640 is off the right side of the screen.
- Excellent! Take two seconds and write another example for a coordinate that is off the screen on the right, circle what changes, and write your definition.
- *Pencils down. Excellent.*
  We've written a function to check whether an object has run off the right side of the screen! But do we care whether the ruby goes off the right side? It's moving to the left! Take two minutes to finish the next design recipe page for `off-left?`
- Now we have a way to check whether something has gone off the right or the left of the screen, but we still haven't told our game what to do when it does. Think back to our dog, in Ninja World. Suppose we'd like it to reappear on the left-hand side. In that situation, what would the next `dogX` be after 640? ZERO.
- We want to change `update-world`, so that it behaves the old way most of the time...but it sets `dogX` to zero when `dogX` is greater than 640. What can we use that makes a function behaves one way for some inputs, but another way for different inputs? Our old friend: `cond`!
- We know that we'll have two conditions right now: when `dogX` is greater than 640, and then the rest of the time. Let's work on the code for this:
-

```
(define (update-world w)
(cond
        [...test...  ...result...]
        [else  (make-world  (+ (world-dogX w) 10)
                               (world-rubyX w)
                               (world-catY w))]))
```

- We still want our original code to be there - it's now going to be used in the else clause, because when `dogX` is not off the right side of the screen we want the world to update normally.
  Walk the students through the transformation.
- Now let's look at the first condition. What is the test that tells us if a number is greater than 640?
- `(off-right (....))`
- But how do we check if the dog is off the right? We'll need to pull the dog out of the world! What function do we use for that? `world-dogX`. So what will our input to `off-right?` be? `(world-dogX w)`
- Let's add this to the definition.
-

```
(define (update-world w)
(cond
        [(off-right? (world-dogX w)) ...result...]
        [else  (make-world  (+ (world-dogX w) 10)
                               (world-rubyX w)
                               (world-catY w))]))
```

- What should we have for our result?

- Well, we know that we need to return a World, since the range of the function is World. That means we can immediately write `(make-world...)`:
-
```
(define (update-world w)
(cond
        [(off-right? (world-dogX w))
                (make-world ...dogX...
                            ...rubyX...
                            ...catY...)]
        [else  (make-world  (+ (world-dogX w) 10)
                               (world-rubyX w)
                               (world-catY w))]))
```

- How should `dogX` change in this condition? We want it to be zero! Since we want to move the dog back to the left side of the screen, we'll just replace it with zero!
- Does `rubyX` change? No. Does `catY` change? No.
-
```
(define (update-world w)
(cond
        [(off-right? (world-dogX w))
                (make-world 0
                            (world-rubyX w)
                            (world-catY w))]
        [else  (make-world  (+ (world-dogX w) 10)
                            (world-rubyX w)
                            (world-catY w))]))
```

- Now let's think about the ruby. Instead of checking if `rubyX` was off the right side of the screen, what would I be checking?
- How do I need to start changing `update-world`? By making a new `cond` branch! Can you walk me through it?
-
```
(define (update-world w)
(cond
        [(off-right? (world-dogX w))
                (make-world 0
                            (world-rubyX w)
                            (world-catY w))]
        [(off-left? (world-rubyX w))
                (make-world (world-dogX w)
                            640
                            (world-catY w))]
        [else  (make-world  (+ (world-dogX w) 10)
                            (world-rubyX w)
                            (world-catY w))]))
```

Tests and Results (Time 45 minutes)

- Now open your own game file.
- The first thing to do is to reformat your `update-world` function so that it uses `cond`, with your current code inside the `else` clause. You have 5 minutes - GO!
- Next copy and paste your `off-left?` and `off-right?` functions from Ninja World into your game. Take one more minute - GO!
- *5, 4, 3, 2, 1...MONITORS OFF!*
- Think about the things in your game that fly offscreen. Do they fly off the left? The right? The top or bottom? Do you need to write `off-top?` or `off-bottom?`
- On Page 34, make a list of the tests that you need to do in the left hand column, to decide whether each thing flies offscreen. For example, with the dog we said `(off-right? (world-dogX w))`. On the right, figure out which world you need to make, so that the thing you're testing re-appears on screen once it's flown off.
- *Work in small groups to complete the workbook page.*

Branches in update-world (Time 15 minutes)

- Look at our `cond` example on the board, for Ninja World. Notice that for each branch, we need a test and a result. This is exactly what you've written in your workbook. All you need to do now is surround each row of your table with square brackets, and type it into your game.
- *Adapt `update-world` so that that each thing re-appears on screen once it's flown off.*

Closing (Time 5 minutes)

- *Have the students show each other their games!*
- *Cleanup, dismissal.*