Unit 4
Welcome to the World

Unit Overview

Students return to the Ninja World game, and codewalk through the 'update-world' and 'draw-world' functions. Making minimal changes to these functions, they are able to modify the dog's speed, add static clouds, etc. They then modify the world to include the ruby's x-coordinate, and systematically update each function in the source code to accommodate this new world. Additional iterations are possible if time allows, by adding more sets of coordinates to the World. Students brainstorm their videogames, and derive the structure for their game world.

Agenda

5 min	Review
20 min	Codewalking
15 min	Extending the World
15 min	Game Brainstorming
10 min	Game Design!
20 min	Defining the World
5 min	Closing

Learning Objectives:

- Deepen their understanding of structures, constructors and accessors by being introduced to a third data structure.
- Discover the event-based microworld implementation of Racket, which uses events to modify the world.

Product Outcomes:

- Students will modify draw-world to add clouds and a ruby
- Students will modify a simple update-world function to change the dog's speed
- Students will iteratively expand the World structure, and trace these changes throughout their program
- Student will define their World structures

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 2 file [NW2.rkt from source-files.zip | WeScheme] preloaded on students' machines*

Preparation:
- *Language Table Posted*
- *Seating arrangements: ideally clusters of desks/tables*

Review (Time 5 minutes)

You guys have gotten really good at structs over the past few weeks, so now it's time for a challenge.
Turn to Page 19 in your workbook. You'll have five minutes to complete the page. Do you think you can do it? Ready? Go!
Be sure to count down the minutes.

Codewalking (Time 20 minutes)

- *MONITORS ON!*
- Open up Ninja World and click "Run". What happens?
- Does it do the same thing we did in our simulation last week? No! Let's walk through it and figure out what's wrong.
- At the top of the screen, you see the DATA section. This is where we define what everything we need to keep track of during the animation.
- As you can see, we have used `define-struct` to define our World structure here. Raise your hand if you can tell me what's in this world struct.
- Next take a look at the section labelled STARTING WORLD. What is `START`? It's a world! How would we get the `dogX` out of the `START` world?
- If the dog is moving ten to the right each time, what should the world be in the next frame? You have ten seconds: Define another world called `NEXT`.
- As you can see, we also defined a bunch of values for images, which we'll use in our game. What are they images of? Type in their names in the interactions window to find out.
- Now that we've got our world structure, we need to know how to draw it. Scroll down to where you see "GRAPHICS". What is the name of this function? `draw-world`. What is the Domain of this function? The Range?
- In our `draw-world` function, `put-image` is placing the `DOG` onto the `BACKGROUND`. What is it using for the dog's x-coordinate? The dog's y-coordinate?
- *Before we move on, I want you to think for a moment about how our simulation last class worked. Every time it ran, `draw-world` had to check what the current world was, for the `dogX`, before it could draw it. Where in `draw-world` is it checking the current world? It's not! We're not looking at the world that we're taking in, so we have no way to change the position of the dog!*
- How would you get the `dogX` out of the world? `world-dogX`. Which world are we going to use? `w`.
- Now...where are we going to put this `(world-dogX w)`? Which number here represents the x-coordinate of the `DANGER` on the `BACKGROUND`?
- Suppose we want to add the `CLOUD`, at the position (500, 400). How could you use `put-image` to stick them on the `BACKGROUND`?
- *Write the code with the kids, since this is their first time with put-image. They'll have time to practice on their own later!*

-
  ```
  ;; draw-world: world -> Image
  ;; place DANGER onto BACKGROUND at the right coordinates
  (define (draw-world w)
    (put-image CLOUD
               500 400
               (put-image DANGER
                          (world-dogX w) 200
                          BACKGROUND)))
  ```
- Click "Run", and take a look at that cloud!
- Now scroll down to where it says UPDATING FUNCTIONS. This code is responsible for changing the World - and in fact we already wrote it!
- What does `update-world` DO to the world? It makes a new world and it adds 10 to the dog's x-coordinate! What does that mean, in terms of how the dog moves? Does it go to the right, left, up, down?
- If the dog is at 100, where will it be next? After that?
- *Take 30 seconds:*
  Write a second example involving your NEXT world, instead of START.
  Count down.
- How could you make the dog move faster? Slower? Backwards?

Extending the World (Time 15 minutes)

- *This section requires that you model each one of the changes to the code on a projector, with students following along. Make sure everyone can see what you're doing!*

- Let's draw our `TARGET` in our world, at (500, 300). What will we need to modify? Our world structure? No - if the `TARGET` isn't moving, then nothing changes, so we don't need to keep track of it in the world. How about `draw-world`? Does that need to change?
- Yes it does! We need to put the `TARGET` image on top of everything we have so far. Add the `TARGET` now, so it shows up when you click "Run". Go!
- Okay, so now suppose the ruby is flying across the screen, moving left slowly at 5 pixels each frame. Now do we need to modify the World? Yes! Why? Because something is changing! Specifically, the x-coordinate of the ruby!
- What do I need to change in my world struct?
- How have our contracts changed? (Call on kids until you get all three (`make-world, world-dogX, world-rubyX`), and write out the contracts for them).
- Okay, let's go through the code, line-by-line, and see what changes. Look at our START variable - does that need to change? Yes! It uses make-world, which now requires two inputs in it's Domain. So what should the ruby's x-coordinate be when the simulation starts? How about 600?
- Now change the definition of `NEXT`. Don't forget to think about how the ruby's x-coordinate has changed!
- Do our image variable definitions need to change? No. Why not?
- What about `draw-world`? Does the contract change? No! The contract says it takes a World as it's Domain, and it still does. All we've changed is what's IN a world. Does `draw-world` still produce an Image? Yes.
- What needs to change about the body of `draw-world`? The ruby's x-coordinate should be pulled from our World structure!
- What about `update-world`? Does the contract change? No! Why?
- Let's get rid of the function body of `update-world` completely, because a lot needs to change here. What do you think we should start out with? Let kids discuss.
- Once again, the contract saves us! Here's a quick tip: if the range is a World, we know that we have to make a world at some point. How do we make a world? `make-world`, of course!
- And the moment we write `make-world`, your instincts should kick in right away. What's in our world? A `dogX` and a `rubyX`...so you should write them down without even thinking! Don't use your brains yet!!!
- NOW you can use your brains. What should our `dogX` be? Well, we still want to move it to the right by 10. How do we do that? 10 plus whatever the dogX is. How do we get the `dogX` out of the world? How would we add ten to that?. And we want to move our `rubyX` left by 5...
- Which world are we pulling the `dogX` and `rubyX` out of?
- Do our examples need to change? Oh, yes they do...
- Look at our first example:
- What's wrong with it? Well, how many things are being passed into `make-world`? Just one: the new x-coordinate. What's missing? The ruby's x-coordinate!
- Okay, and what do we want to do to the x-coordinate? We want the ruby to go left by 5, so what code do I write?
- *Take 30 seconds, and fix the second example. GO!*

Game Brainstorming (Time 15 minutes)

- All of you have been working with structures for the last three lessons, and you've gotten really good at defining, making and accessing them. Today, you're going to define the World structure for YOUR GAMES!
- Suppose I have a racing game, where my player is at the bottom of the screen, sitting in their car. In front of them, I have two lanes, with cars coming at me as I catch up to them. To move out of the way, I need to change into the left or right lane of the road. What are all the things I need to keep track of in my game?
    - PlayerX - a number
    - CarY - a number
    - Car2Y (if I want another car) - a number
    - Score - a number
- How would I define this world?
- How do I get the playerX out of my word? My CarY? My Car2Y? The score?
- What if I wanted the player's car to change color as the score goes up? How would my world structure need to change?
- Now think about YOUR game - what will be changing in your world?
  Pass out some scratch paper for the students to brainstorm on. Make sure you force them to think about their world structures, and start simple!

Game Design! (Time 10 minutes)

- It's time to start work on your game!
- Turn to [Page 20]() in your workbooks. First, you're going to draw a rough sketch of what your game should look like when the user clicks "Run". You'll have 5 minutes - GO!

- *Make sure kids keep it simple! Limit their world struct to five things, initially.*
- Now make a list of all the images you'll need in your game.
- Now make a list of everything that changes in your game - if something moves, will you need to keep track of it's x-coordinate? y? both?

Defining the World (Time 20 minutes)

- Now that you've gotten a list of everything that changes, it's time to turn them into a World structure!
- Turn to Page 21 in your workbooks, and define your world structure. When you're done, write down all of the contracts that you need to work with your structures. Take 10 minutes - GO!
- *Review each team's structure, and make sure it accurately models their world. Also be sure to check their contracts!*
- Define an example world called START, which is how your world should look a split-second after the game begins. Write it in on the bottom of Page 21. You'll have two minutes. Go!

Closing (Time 5 minutes)

- *Let's go around the class and have you each talk about your game. Once you've explained it, tell the class what you have in your World structure.*
- *NOTE TO INSTRUCTORS:*
    - *Make sure student names are on page 20*
    - *Take page 20 itself, or take photos of page 20, to prep game images for the next unit.*
    - *Images should be in PNG or GIF format. Background images should be 640x480, and character images should generally be no larger than 200px in either dimension. Make sure that the character images have transparent backgrounds!*
    - *TIP: use animated GIFs for the characters - not only does the animation make the game look a lot better, but these images usually have transparent backgrounds to begin with.*