Unit 9
Completing Games

Unit Overview

This unit includes instructions for adding frequently-requested elements to students' games, such as extra levels and a scoring system. Students comfortable with structures are encouraged to use nested structures in their games for more complexity.

Agenda

5 min	Introduction
15 min	Randomizing Ninja World
35 min	Scoring and Levels
30 min	Challenge: Nested Structures
5 min	Closing

Learning Objectives:

- Reinforce understanding of structures as they are used in their games

Product Outcomes:

- Students will use the random function to make their game characters appear at different loations on the screen
- Students will add a scoring system to their games
- Students will add levels to their games
- Students will use nested structures to add complexity to their games

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*
- *Language Table*
- *The Completed Ninja World file [NWComplete.rkt from source-files.zip | WeScheme] preloaded on students' machines*
- *New background image for Ninja World level two [bg2.jpg from source-files.zip or your own 640 x 480 image*

Preparation:

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

Introduction (Time 5 minutes)

- Congratulations! You've worked very hard throughout the last 8 lessons, and now your games are almost completed!
- Your player can move any way you want, you've written the code to detect how far characters are from each other, and you've defined a whole bunch of conditional cases for different actions in your game.
- Have you brainstormed some things you want to add to your game? You have 5 minutes to discuss your ideas with your partner, and talk about how you could add them.
- *At this point in the course, students will have very different games, and will probably need individual help adding the finishing touches or extra elements. This unit includes ideas and instructions for frequently requested game elements (Using Ninja World as a template), but feel free to have your students get creative with their additions! For more practice with algebra, try challenging students to some harder problems.*

Randomizing Ninja World (Time 15 minutes)

- If you open up your Ninja World file, you'll see our (almost) completed game! However, right now the ruby and dog appear at the same part of the screen every time, making this a pretty easy game. What will the y-coordinate of the dog always be? What about the ruby?
- Instead of appearing at the top of the screen every time, what if we could make the dog show up at a random y-coordinate every time it goes off the screen?
- Racket already has a function to give us a random number, which could represent a character's y-coordinate. `random`! random takes in one number as it's domain, and returns a random number between 0 and that number. So if I write `(random 480)` in my code, it could give me back any number between 1 and 480.
- If we want the y-coordinates of our dog to change, we'll have to add it to our world structure. Go back to the top of the page where we defined our world and add in a `dogY`. Don't forget to redefine your START and NEXT worlds, to account for the extra item in your world struct!
-
```
;; The World is the x and y positions of the dog, x position of the ruby,
;; and y position of the cat
(define-struct world (dogX dogY rubyX catY))
```

- What function draws the dog on the screen with the rest of the game characters? `draw-world`. Right now this function draws the dog at it's current x-coordinate, and a pre-set y coordinate. How do we get the dog's y-coordinate out of the world? `(world-dogY w)`. Change the `draw-world` function so that it draws the dog at the current y-coordinate instead of 400.
- We said we want our dog's y-coordinate to change when it leaves the screen. What function changes the game state depending on the game's conditions? `update-world`!
- Our first cond branch in update-world checks whether the cat collides with the dog. If this happens, we don't want the dog to stay at its current y-coordinate. We already have the dog reappearing on the left side of the screen (by setting its x-coordinate to -50). Let's reset its y-coordinate to a random number between 0 and 480. Do you remember how to do this?
-
```
[(collide? 320 (world-catY w) (world-dogX w) (world-dogY w)) (make-world -50
                                                             (random 480)
                                                             (world-rubyX w)
                                                             (world-catY w))]
```

- Further down in `update-world`, we check to see if the dog has gone off the right side of the screen. Once again, we want to make the dog reappear at a random y-coordinate!
- Carefully go through your code- since we changed our world structure to include a `dogY`, we'll need to make sure we're including it every time we call `make-world`!
- Once the dog is reappearing randomly when it leaves the screen, you can make the same changes to the ruby's y-coordinate to make it appear randomly, or add this concept to your own game.

Scoring and Levels (Time 35 minutes)

- We've got Ninja World looking great! But right now there's not a lot of variety. The player avoids the dog and gets the ruby over and over again. We should mix things up a bit: how about adding new levels?
- Typically a game would progress if the player has reached a certain goal. (Collected a certain number of rubies, destroyed a certain number of zombies, or reached a certain score). Let's start by adding a scoring system to our Ninja World game.
- The score is something that will be changing in the game, So we know it has to be added to our world structure. What data type is the score? What will the score in our START world be?
-
```
;; The World is the x and y positions of the dog, x position of the ruby,
;; y position of the cat, and the player's score
 (define-struct world (dogX dogY rubyX catY score))
```

- Remember! Since we're changing the world structure, we now have to go through our game code- every time we make a

world, we need to include the world's score.

- How to we get the score out of our world? `(world-score w)`. Take 5 minutes and add a score to your game, every time `make-world` is used.
- Now that we have a score, when should it increase or decrease? In Ninja World, I want the score to go up by 30 points when Ninja Cat collides with the target, and down by 20 points when colliding with the danger. Which of our cond branches in `update-world` checks these conditions?
- If the player collides with the danger, we want to make a new world with a lower score. All we have to do is subtract 20 from the score when we call `make-world`.
-

```
(define (update-world w)
(cond [(collide? 320 (world-catY w) (world-dogX w) (world-dogY w)) (make-world -50
                                                                   (random 480)
                                                                   (world-rubyX w)
                                                                   (world-catY w)
                                                                   (-(world-score w) 20))]))
```

- On the next cond branch, make the score increase by 30 points when the cat collides with the ruby.
- Our game has a scoring system! Now let's add some levels.
- When the player progresses to level two I want the game to have a different background image. The player reaches level two when his or her score is greater than 500.
- Let's think about the first part- where do we define our BACKGROUND image? We want to keep our original background for the first level, but let's define a new variable, BACKGROUND2, that will be used for level 2.
  You can use the provided background image, or walk students through finding and adding their own image to the game.
- Now that we have another background image, we need to draw it in our game- but we only want to see this new background when a certain condition is met. When will the player reach level 2? When their score is greater than 500.
- We'll need to change our draw-world function so that it uses cond! Leave the current code alone for now and start right under `(define (draw-world w))`. What's the first thing we write? cond! And what's the first condition that we're checking? Whether the world's score is greater than 500!
-

```
(define (draw-world w)
(cond
[(> (world-score w) 500) (.....)]))
```

- If the world's score is greater than 500, the player progresses to the next level. For now, the only thing that I want to change in level two is the background image. The second part of this cond statement will look similar to the code you already have for draw-world, starting with `put-image`. What needs to change? Instead of putting all your images on top of BACKGROUND, you'll put them over BACKGROUND2, your new background image.
-

```
(define (draw-world w)
(cond
[(> (world-score w) 500) (put-image PLAYER
                           320 (world-catY w)
                               (put-image TARGET
                               (world-rubyX w) 300
                                     (put-image CLOUD
                                      500 400
                                         (put-image DANGER
                                         (world-dogX w) (world-dogY w)
                                               BACKGROUND2))))]))
```

- Don't forget to add an 'else' case before your original code, right underneath what you just wrote. If the score is **not** greater than 500, the world will be drawn with the images on the original background:

```
[else
(put-image PLAYER
 320 (world-catY w)
        (put-image TARGET
        (world-rubyX w) 300
              (put-image CLOUD
               500 400
                   (put-image DANGER
                   (world-dogX w) (world-dogY w)
                          BACKGROUND))))]
```

- Great! Now our game has a level 2! You can use the same process to add more levels when the score gets even higher.
- *Some more options for students who finish early:*
  - *Change the update-world function so that the danger and target move faster if the score is greater than 500.*
  - *Use the text function to display a game over message on the screen when the score drops below 0.*
  -

Challenge: Nested Structures (Time 30 minutes)

- Now that you know about data structures and how to use them, you can have games that are even more complex. For example, you could make a game with many different characters by making each character their own data structure!
-
```
;; A character is an image, speed, x-coordinate, and y-coordinate.
(define-struct character (image speed x y))

(define DOG (make-character DANGER 10 -50 400))
(define CAT (make-character PLAYER 20 320 240))
(define RUBY (make-character TARGET 10 690 300))
```

- Take a few minutes with your partner and define some characters for your own game. They don't have to follow the same pattern- your characters can have a health property instead of a speed, for example. Just be sure your newly defined characters have the same properties as the character structure you define!
- If each character is now its own structure, what will our world look like? Something like:
-
```
;; The World is three characters: (dog, cat, ruby) and a score
(define-struct world (c1 c2 c3 score))
```

- Why is it important to use variable names (c1, c2, and c3) instead of just defining our world struct to include specific characters (DOG, CAT, RUBY)? The same reason why we use variables in functions: we want to be able to change the value of those characters later if need be.
- When we define our first world, we can then make our predefined character structs part of that world.
- `(define START (make-world DOG CAT RUBY))`
- We need some way to access parts of a struct, even if it's inside another. This is just like accessing any part of a struct. How do I get the first character out of my starting world? `(world-c1 START)`.
- What does this expression evaluate to? A character struct! How would I get the speed out of this character? `(character-speed (world-c1 START))`
- For even more of a challenge, you can make every level in your game its own structure. A level could have a background image, characters, a boolean value representing wheather the player has collided with another character, or anything you like!

Closing (Time 5 minutes)

- *Congratulations! You've completed every lesson, and your games look fantastic! You've all been really impressive, and it's a pleasure working with you. We hope you'll take these games home and keep hacking! Keep learning!*
- *Have students show each other their completed games!*