# Unit 6
# Key Events

## Unit Overview

Students return to the subject of partial functions, this time defining a key-event handler that modifies their world when certain keys are pressed.

Learning Objectives:

- Extend their understanding of events to cover key-events
- Deepen their knowledge of conditionals, by combining them with struct accessor and constructor functions.

Product Outcomes:

- Students will define keypress, and hook it up to the event handler

**Agenda**

| | |
|---|---|
| 15 min | Introduction |
| 35 min | Keypress |
| 35 min | Keypresses in your Game |
| 5 min | Closing |

**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*
- *Signs for kids, entitled "update-world", "draw-world" and "big-bang"*
- *Cutout images of the dog and ruby*
- *The Ninja World 4 file [NW4.rkt from source-files.zip | WeScheme] preloaded on students' machines*

*Preparation:*

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

## Introduction (Time 15 minutes)

- *On the projector, have the code for Ninja World.*
- *Draw a box on the board with* `(make-world 0 640)` *in it, labelled "world". Ask for a volunteer, and hang the update-world nametag around their neck.*
- You're going to be `update-world`. What is your contract? And what do you do? Excellent. This box is your world. If I were to say `(update-world (make-world 0 640))`, what would you change about this world?
  *The student should erase the 0 and write a 10, and erase the 640 and write a 635. If they are stuck, refer to the code.*
- Let's practice this a few times: "update-world this world"...20 630..."update-world this world"...30 625. Excellent.
- Now I need another volunteer.
  *Take one student, and put the big-bang sign around them.*
  Your name is `big-bang`. You're sort of like the quarterback here: you start the whole animation, and you have a timer. The class will yell "tick!" every five seconds, and you're going to tell `update-world` to update the world, just like I did. Let's try it out - every five seconds, I want you to give the current world to `update-world`, who will then update it and replace it with the new world. Ready? GO!
- *Let this go on for a few iterations, so the rest of the class can see the world structure being changed while they count down.*
- Which other function do we have to use to see an image of the world?
- I need another volunteer!
  *Take one student, and put the draw-world sign around them.*
  You guessed it! Your name is `draw-world`. Whenever you are evaluated, you'll be given a World and you'll have to place the image of the dog and the ruby at the appropriate spots.
- Change the value of the world back to 0 640. "draw-world this world"!
  *The student should look at the new value, and move the dog and ruby to the right spots. Repeat a few times, with different values for the World.*
- Okay, now let's put it all together!
  *Return the World value to 0 640.*
  `big-bang`, I want to call out "update-world this world" and "update-world this world". Try it out!
  *Make sure the World is updated to 10 595, the dog is drawn at 10, and the ruby at 595.*
  What's the value of our world now? 10 595! So what do you do? You call out "update-world this world" and "draw-world this world".
- When I say GO, the class will call out "tick" every five seconds. `big-bang`, what will you do every time you hear that?
  *Let big-bang explain.*
  Okay - let's do it!
- Kids see the dog moving across the screen, as a result of the World Structure changing.
- Excellent! We've made both the dog and the ruby move! But that's old news. We've already done that sort of thing, with the stuff moving on its own, in our games! Let's add to it.
- Say that I wanted to add a Ninja Cat. The time is nigh: we need to get that ninja onto the screen, so that our game is playable! Ninja cat is going to be able to move up and down. Do we need to change anything to make this work?
- We need to keep track of our catY, so we need to add it to our world. Where's the part of the code where we defined our struct? `define-struct`.
  *Change it in the code.*
- Excellent. So what does that mean about our simulation? Do we have to change anything here? Yes! The world in our simulation doesn't match! Let's add another part to our world struct...and what type is it going to need to be? A number.
- I want Ninja Cat to be right in the center of the screen. What does that make his y-coordinate? 240. `update-world`, can you change the world to have another space, starting at 240? `draw-world`, where are you going to put the cat? `big-bang`, does your job change? Nope! Alright...let's play the game.
  *Let a few iterations pass...*
  The cat doesn't move!
- Okay, now I'm going to hit the "up" key. Wait, why are you moving the cat? Which function are you? Is that your job? No...it's not in the code!
- Right now, even though we're putting the cat into the game, we don't have a function to take in keypresses and make it work! This is what we're going to write next.
- *Thank the volunteers, and let them have a seat*
- Before we figure out how to write the function to move the cat...we need to actually add him into the game! Which functions will we need to change?
- We need to change our `define-struct` statement, to add catY. We need to change every single `make-world`. We need to change `update-world`, and `draw-world`...so just about everything!

## Keypress (Time 35 minutes)

- We're going to use the Design Recipe to write a function `keypress` for Ninja World. Take a look at Page 28 in your workbook. What's the first step in the Design Recipe?
- Step 1 - Contract and Purpose Statement
  - What's the Name of our function? keypress.
  - What about the Domain? What do we need to know in order to handle a keypress? Well, we need the World - otherwise we wouldn't know what to update! But we also need to know what key was pressed. What are `"up"` and `"down"`? A number, a string, a boolean or an image? We need a World and a String.
  - The Range? A world.

  - What's a good purpose statement for this function?
    *Let students discuss.*
- Step 2 - Examples
  - Let's make an example using our START world, when the user presses "up":

  -
    ```
    (EXAMPLE (keypress START "up")
              ...)
    ```
  - What should we get back? Well, we know that our Range is a World, so immediately we can write:

  -
    ```
    (EXAMPLE (keypress START "up")
             (make-world ...dogX...rubyX...catY))
    ```
  - Does the dogX change when the user types "up"? No! So how do we get the old dogX out of the START world? (world-dogX START). So now we have:

  -
    ```
    (EXAMPLE (keypress START "up")
             (make-world (world-dogX START)...rubyX...catY))
    ```
  - Does the rubyX change when the user types "up"? No! So how do we get the old rubyX out of the START world? (world-rubyX START). So now we have:

  -
    ```
    (EXAMPLE (keypress START "up")
             (make-world (world-dogX START) (world-rubyX START) ...catY...))
    ```
  - How does catY change? We need to add 10 to the old catY.

  -
    ```
    (EXAMPLE (keypress START "up")
             (make-world (world-dogX START) (world-rubyX START) (+ (world-catY START) 10)))
    ```
  - Now how would I make an example using the "down" key?

  -
    ```
    (EXAMPLE (keypress START "down")
             (make-world (world-dogX START) (world-rubyX START) (- (world-catY START) 10)))
    ```
- Step 3 - Definition
  - What goes in our function header? This one is pretty straightforward:

  -
    ```
    (define (keypress w key)
             ...)
    ```
  - What now? This is a test of your programming intuition...we have two different examples here, where we add 10 in one case but subtract 10 in another. How can a function behave so differently? It has multiple conditions, with a different response to each. You've actually seen this before, back in Bootstrap I....COND!
  - So we start with cond, and use those square brackets to add a branch. We know that every branch has a test and a result.

  -
    ```
    (define (keypress w key)
       (cond
             [...test...   ...result...]))
    ```
  - Let's start with the "up" branch. We need to test if "key" equals "up". How do we check if two strings are equal?

  -
    ```
    (define (keypress w key)
    (cond
            [(string=? key "up")   ...result...]))
    ```
  - So what's the result, if the key is "up"? Well, we can look back at our examples for help! Just copy in the example we made for "up", and change that START to a w:

  -
    ```
    (define (keypress w key)
    (cond
            [(string=? key "up")   (make-world (world-dogX w)
                                               (world-rubyX w)
                                                  (+ (world-catY w) 10))]))
    ```

- What about the "down" key? Go ahead and write the down branch yourself.

-
  ```
  (define (keypress w key)
  (cond
          [(string=? key "up")   (make-world (world-dogX w)
                                             (world-rubyX w)
                                                (+ (world-catY w) 10))]

          [(string=? key "down")   (make-world (world-dogX w)
                                               (world-rubyX w)
                                                 (- (world-catY w) 10))]))
  ```
- What about any other key? Should the world change if the user hits the spacebar, or the "r" key? No. So we add add "else" branch, which returns the same world that was passed in.

-
  ```
  (define (keypress w key)
  (cond
          [(string=? key "up")   (make-world (world-dogX w)
                                             (world-rubyX w)
                                               (+ (world-catY w) 10))]

          [(string=? key "down")   (make-world (world-dogX w)
                                               (world-rubyX w)
                                                 (- (world-catY w) 10))]
          [else w]))
  ```

- *If students finish early, have them add a keypress for "c", which causes the cat to jump to the center, or any other "cheat codes".*

## Keypresses in your Game (Time 35 minutes)

- In Ninja World, what keys could the user press? "up" and "down". What field in the World changes when the player presses "up"? catY. What does it change by?
- Think about the user playing your game for a minute. How will they control the game? What key will make YOUR player move up? Down? What else can they do?
- Turn to Page 27 in your workbooks. Write down the various keys that the user can hit to control the game. For each one, make sure you write down the field in your world struct that changes, and how it changes! You have 2 minutes. GO!
- Now turn to Page 30 in your workbooks. Choose 3 keys that control the game, and write the examples of what should happen to your START world in each of them.
- *Once you're done with the design recipe, you can turn your monitors on and start writing the keypress function into your games.*

## Closing (Time 5 minutes)

- *Have students show each other their controllable games!*