Scoring

Students extend the data structure that represents their game to include a score, then modify their helper functions and event handlers to update and display that score.

Product Outcomes	• Students add a `score` field to their `gameState` structure • Students modify their `draw-state` function to display the score on the screen • Students modify other parts of their code to increment or decrement the score
Materials	• Slides are not yet available for this lesson • Printable Lesson Plan (a PDF of this web page)
Prerequisites	• Simple Data Types • Contracts • Simple Inequalities • Piecewise Functions and Conditionals • Compound Inequalities: Solutions & Non-Solutions • Introduction to Data Structures • Structures, Reactors, and Animations • Key Events • Build Your Own Animation

Glossary

helper function :: a small function that handles a specific part of another computation, and gets called from other functions

Adding a Scoring System *45 minutes*

Overview

Students add a score to their game.

Launch

The score is something that will be changing in the game, so you can be sure that it has to be added to the `____State` data structure. In our example Ninja Cat program, we've called our structure `GameState`, which currently contains the x and y-coordinates for our player, danger, and target, plus the speed of the danger, and speed of the target. Your game(s) will likely have different structures.

Investigate

- What data type is a score? Number, String, Image, or Boolean?
- What would be the score in your starting game state? (we called this `START` in our game.)
- Change the data structure in your game so it includes a score.

Remember: Since your structure is changing, you now have to go through your game code — *every* time you call the constructor function for your structure (ours is `game()`), the score must be included. It may be helpful to add the score as the very first or last field of the structure, to make this easier.

How would you get the `score` out of one of your instances?

The `GameState` structure for our Ninja Cat game now looks like this:

```
data GameState:
    game(
      playerx :: Number,
      playery :: Number,
      dangerx :: Number,
      dangery :: Number,
      dangerspeed :: Number,
      targetx :: Number,
      targety :: Number,
      targetspeed :: Number,
      score :: Number)
end
```

Reminder

Your students will likely have radically different games at this point in the course. This lesson is not meant to be followed exactly, but rather used to give students an idea of what steps they should take to add a scoring system to their own games. For extra practice, students can work through adding a scoring system to the Ninja Cat program as well as their own games.

Now that the game has a score, that score needs to actually increase or decrease depending on what happens in the game. For our Ninja Cat game, we'll say that the score should go up by 30 points when Ninja Cat collides with the ruby (target), and down by 20 points when she collides with the dog (danger).

- Which of the `if` branches in your `next-state-tick` function checks whether your player has collided with another character?
- How would you decrease the game's `score` by 20 points if the player collides with the danger?
- Hint: How many dangers does your game have? If there are multiple things your player could hit to lose points, remember to check for each possible collision condition!

If you completed the optional challenge at the end of the Collisions Feature to write the function `game-over`, you already have your own ***helper function*** to check whether or not your game over condition is met. That will be the first condition inside `next-state-tick`, since we don't want the game to continue if it's already over! (In our Ninja Cat game, game-over returns true if the cat collides with the dog, AND the cat is on the ground.) After checking whether or not the game is over, the next three conditions in our `next-state-tick` function check whether the player has collided with the danger and target, as well as whether the player is jumping on the danger:

```
# next-state-tick :: GameState -> GameState
fun next-state-tick(g):
  if game-over(g): g
  # if player and danger collide while player is on the ground,
  #reset player and danger and decrease score
  else if is-collision(g.playerx, g.playery, g.dangerx, g.dangery)
    and (g.playery < 110):
    game(
      START.playerx,
      START.playery,
      750,
      g.dangery,
      g.dangerspeed,
      g.targetx,
      g.targety,
      g.targetspeed,
      g.score - 20)
    # if player and danger collide while player is jumping,
    # reset danger and increase score
  else if is-collision(g.playerx, g.playery, g.dangerx, g.dangery)
    and (g.playery > 110) and (g.playery < 300):
    game(
      g.playerx,
      200,
      -100,
      0,
      0,
      g.targetx,
      g.targety,
      g.targetspeed,
      g.score + 30)
  # if player and target collide, reset target and increase score
  else if is-collision(g.playerx, g.playery, g.targetx, g.targety):
    game(
      g.playerx,
      g.playery,
      g.dangerx,
      g.dangery,
      g.dangerspeed,
      -400,
      0,
```

```
      0,
      g.score + 30)
```

Change your own game code so that your score increases and decreases depending on various game conditions: Maybe your score increases when the player collides with a target, reaches a specific area of the screen, or reaches a specific area *only after* picking up an item. Maybe your game's scoring system isn't a seprate score at all, but a timer that increases every tick, and represents how long someone has been playing your game. There are lots of ways to implement a scoring system, and which one you choose will depend on the specific mechanics of your individual game.

Now your scoring system is in place, but how will the person playing your game know what their score is? You'll want to display the score on the screen.

Which function handles how the game state is drawn?

In the `draw-state` function, images are placed onto the background using `put-image` to draw the game. But the score is represented by a Number: we need a way to represent it as an Image. Thankfully, Pyret has some built-in functions that can help with this: the function `num-to-string` takes in a Number for its domain and returns a String representation of that number. This string can then be passed to the text function to return an Image that can be used in `draw-state`.

Copy the following contracts into your workbook:

- `# num-to-string :: Number -> String`
- `# text :: String, Number, String -> Image`
- How would you use the `num-to-string` and `text` functions together to draw the score into the game?
- How do you get the `score` out of the game state?
- How large should the text of the score be? Where should it be placed on your game scene?

The expression:

```
put-image(text(num-to-string(g.score), 20, "white"), 320, 240,
BACKGROUND-IMG)
```

will place the score (drawn in size 20 white text) onto the center of the BACKGROUND-IMG.

Use these functions to draw the score onto your game screen. You could also use the string-append function to make it clear to players that the number they see is their score, like so:

```
text(string-append("Score: ", num-to-string(g.score)), 20,
"white")
```