DEVELOPER'S GUIDE

(WORD COUNT: 971 words)

Excluding all the clipped codes, function names and titles.

Structure of the code file:

Our main code file is structurally divided into 4 main sections of code. The code is separated into "BACKGROUND DETAILS", "GAME-SCREEN LOGIC AND MECHANICS", "LOADING SCREEN", and "GAME-OVER SCREEN". All the sections contain the information relevant to the title.

Furthermore, all our functions are given a tag either "BACKGROUND", "DORAEMON", "LOGIC", or "MECHANICS". As it suggests, "BACKGROUND" and "DORAEMON" tags are given to functions related to its name. "MECHANICS" tags are given to functions which are responsive to the user's inputs or which have visual output such as **(ending-screen gui)**. "LOGIC" tags are given to information processing functions like **(tick gui)** functions.

As it can be seen, it is possible to easily guide through our code as it is divided in sections while being named very sensibly as well as having specific tags to specify the code's functionality.

Data Types:

Our program has one main struct "GUI" **(Graphical User Interface)** which contains 3 Numbers, 2 POSNs, 3 Booleans and 1 struct called ItemDetails. GUI holds y-coordinate **(current-state)** of DORAEMON, coordinates of 2 obstacles **(obstacle, obstacle1),** boolean state of indicating if DORAEMON crashed **(dead)**, score of the player **(score)**, boolean state indicating if the game is being played **(game)**, game-over timer **(timer)**, boolean state indicating if the application is running **(quit?)**. ItemDetails contain the coordinate of the item **(cord)**, the duration of the effect **(time)**, distance between the pillars **(pillar)** and the type of effect it is having **(type).**

<u>STARTING SCREEN (From Line 702):</u>

Main features of "Starting screen" are functions **(loading-screen gui)** and **(tips)**. **(tips)** take a list of tips stored in "TIPS" and randomly display one of them by checking the value of randomly generated number "Z" (between 1 to 4 as there are 4 tips) as shown below:

```
(define Z (random 1 5))

(define tips ;; RANDOMLY CHOOSES ONE TIP FROM THE LIST
  (cond
    [(= Z 1) (first TIPS)]
    [(= Z 2) (second TIPS)]
    [(= Z 3) (third TIPS)]
    [else (fourth TIPS)]))
```

Function **(loading-screen gui)** receives various static values and also the output of (tips) and compiles them together to build the starting page using **(place-image)** function from the **(require 2htdp/image)** library as shown below:

```
(define (loading-screen gui)  ;; STRUCTURE FOR LOADING SCREEN IN THE BEGINNING
  (place-image LOGO 520 200
             (place-image (msg gui) 520 400
                        (place-image (bubble gui) 520 500
                                   (place-image DORAEMON 100 160
                                              (place-image QUIT 200 560
                                                         (place-image DAY 520 300 BACKGROUND)))))))
```

GAME-OVER SCREEN (From Line 812):

In the code for "GAME-OVER SCREEN" there are 4 main functions to pay attention to which are (end gui), (game-over gui), (ending-screen gui), and (handle-mouse) functions.

First of all, **(ending-screen gui)** function has similar function and structure as (loading-screen gui) as it is the compiler used to build the game-over screen. Secondly, (end gui) determines the state of the game by checking if DORAEMON's y-value is between 0 to 600 and if it is not it will stop the game and load the ending screen.

```
(define (end gui)
  (cond
    [(or (> (GUI-current-state gui) 600) (> 0 (GUI-current-state gui)))
     (... GUI ...)]))
```

Chronologically, the **(game-over gui)** function will check the score obtained by the user and send the corresponding message to the user including their score.

```
(define (game-over gui)
  (cond
    [(and (boolean=? (GUI-dead (end gui)) #t)
          (boolean=? (GUI-game (end gui)) #f)
          (... GUI-score ...))
     (... GUI ...)] …))
```

Lastly, "GAME-OVER SCREEN" uses a mouse-handling function associated with **(require 2htdp/universe)** called **(handle-mouse)** to employ the "RETURN" button which leads the user back to the starting screen so they can play the game again. The mouse-handling function checks the location of the click and if it is on “RETURN”, it will reset the (GUI) struct to initial state and render the starting screen.

```
(define (handle-mouse gui x-mouse y-mouse mouse-event)
 (cond
   [(and (boolean=? (GUI-dead gui) #t) (boolean=? (GUI-game gui) #f)
       (string=? "button-down" mouse-event)
       (> 1020 x-mouse 900) (> 575 y-mouse 545))
    (make-GUI 300 (make-posn 500 (random -200 150))
           (make-posn 1000 (random -200 150))
           #f 0 #f 0 #f I1)]
   [else gui]))
```

<u>BACKGROUND DETAILS (From Line 22)</u>

Most of the “BACKGROUND DETAILS” consist of constant variables such as background images **“NIGHT”**, **“DAY”** and etc. Also **“TUBES”** are defined here using **(require 2htdp/image)** library to reduce lag as rendering multiple external images in *DrRacket* at the same time is difficult. External images are imported using:

```
(define IMAGE_NAME (bitmap “file name”))
```

The **(theme gui)** function changes the background image between **“day mode”** and **“night mode”** by checking the (GUI-score) using every 1000 points using “cond”.

```
(define (theme gui)
 (cond
   [(= (remainder (quotient (GUI-score gui) 1000) 2) 0) DAY]     ;; Night Mode
   [(= (remainder (quotient (GUI-score gui) 1000) 2) 1) NIGHT]))  ;; Day Mode
```

(character gui) is more complex as it includes “Easter egg” where the DORAEMON turns into another character “NOBITA” (character from same animation) when the score reaches 2112 (birth year of DORAEMON) for 93 ticks (DORAEMON is born on September 3rd). Otherwise DORAEMON changes image when it crashes.

```
(define (character gui)
  (cond
    [(and (>= (GUI-score gui) 2112) (>= 2205 (GUI-score gui))) EASTER]
    [(boolean=? (GUI-dead gui) #f) DORAEMON]
    [else CRASH]))
```

GAME-SCREEN LOGIC AND MECHANICS (From Line 200)

This section has most number of significant functions starting from **(respawn gui), (bottom-pillar gui), (x-detector gui), (y-detector gui), (collision-detector gui), (buff-location), (buff-collision gui), (render gui), (tick gui)** and also **(key-handler).**

Logically, **(x-detector gui), (y-detector gui)** and **(collision-detector gui)** are the first to be understood. (x-detector gui) checks if the DORAEMON’s x-coordinate is in the range of the pillar and returns corresponding boolean expression. The (y-detector gui) does the same thing for y-coordinate values. Finally, (collision-detector gui) checks if the boolean returned from x and y-detector functions are “TRUE” and if both are true it changes the (GUI-dead) to true.

```
(define (x-detector gui)
  (cond
    [(and (> 168 (posn-x (GUI-obstacle gui)))
         (> (posn-x (GUI-obstacle gui)) 32))
     #t]
    [else #f]))

(define (y-detector gui)
 (cond
   [(and (> (GUI-current-state gui)
            (+ (posn-y (GUI-obstacle gui)) 254))
         (> (- (bottom-pillar gui) 254)
            (GUI-current-state gui)))#f]
   [else #t]))

(define (collision-detector gui)
 (cond
   [(or (and (boolean=? #t (x-detector gui))
         (boolean=? #t (y-detector gui)))
        (and (boolean=? #t (x1-detector gui))
         (boolean=? #t (y1-detector gui))))
    (make-GUI (GUI-current-state gui) (GUI-obstacle gui) (GUI-obstacle1 gui) #t (GUI-score
gui) #t (GUI-timer gui) #f (GUI-item gui))]
    [else gui]))
```

The code file contains a brief description of why those numerical values were used.

When the DORAEMON doesn't collide with the pillars **(respawn gui)** function will check if the x-coordinate of pillars reaches 0 and if it does it will reset the coordinate of the obstacle back to 1000.

```
(define (respawn gui)
 (cond
   [(... (GUI-obstacle gui) ...)
    (...)]
   [(> 0 (posn-x (GUI-obstacle1 gui)))
    (make-GUI ... (make-posn 1000 (random -200 150)) ...)]
   [else gui]))
```

There are functions called **(x-buff gui)** which carry out a similar role as (x-detector) and naturally, it precedes this feature to **(y-buff gui)** as well. In case of (buff-location gui) unlike (collision-detector), if it doesn't come into contact with the DORAEMON it will respawn the item at a random coordinate when x-coordinate of the item reaches 0, and if it comes in contact, it will set the **(ItemDetails-time (GUI-item gui))** to 1 from 0.

```
(define (buff-location gui)
  (cond
    [(> 0 (posn-x (ItemDetails-cord (GUI-item gui))))
     (... GUI ...)]
    [(and (boolean=? #t (x-buff gui))
          (boolean=? #t (y-buff gui)))
     (... GUI ...)]
    [else gui]))
```

When the item is consumed the distance between pillars will constantly change according to function **(buff-collision gui)** because bottom pillar's coordinates are determined in relation to coordinates of top pillar (GUI-obstacle) and the gap which is determined by (buff-collision):

```
(define (bottom-pillar gui)
  (+ (posn-y (GUI-obstacle gui)) 450
     (ItemDetails-pillar (GUI-item (buff-collision gui)))))
```

(buff-collision gui) randomly chooses the gap of the pillars between either 170 or 270 every 130 ticks when DORAEMON comes in contact with the item, otherwise, the gap will be 220.

```
(define (buff-collision gui)
  (cond
    [(= (ItemDetails-time (GUI-item (buff-location gui))) 131)
     (make-GUI ... (make-ItemDetails ... 220 ...))]
    [(and (and (> 131 (ItemDetails-time (GUI-item gui)))
             (> (ItemDetails-time (GUI-item gui)) 0))
         (= (ItemDetails-type (GUI-item gui)) 1))
     (make-GUI ... (make-ItemDetails ... 170 ...))]
    [(and (> 131 (ItemDetails-time (GUI-item (buff-location gui))))
             (> (ItemDetails-time (GUI-item (buff-location gui))) 0)
         (= (ItemDetails-type (GUI-item gui)) 2))
     (make-GUI ... (make-ItemDetails ... 270 ...))]))
```

The **(render gui)** function checks **(GUI-dead)** and **(GUI-game)** boolean value to determine which screen to draw. If both are false, it draws STARTING SCREEN; if (GUI-dead) is false but (GUI-game) is true, it draws game screen; if both are true, it draws animation of "CRASH" falling at a constant speed which will lead to (end gui) function and hence to GAME-OVER screen.

```
(define (render gui)
  (cond
   [(and (boolean=? (GUI-dead (collision-detector gui)) #f)
        (boolean=? (GUI-game gui) #f)) (loading-screen gui)]
   [(and (boolean=? (GUI-dead gui) #t) (boolean=? (GUI-game gui) #f))
    (ending-screen gui)]
   [else (place-image (SCORE gui) 850 90
                  (place-image ITEM
                           (posn-x (ItemDetails-cord (GUI-item gui)))
                           (posn-y (ItemDetails-cord (GUI-item gui)))
                           (place-image (character gui)
                                    X (GUI-current-state gui)
                                    (pillars gui))))]))
```

More specific description will be shown in the code file.

The **(tick gui)** is constantly run and used as the medium to control the whole application. It affects the location of all objects in the game and it is used to measure the score, and time. It performs different actions according to the conditions given.

CODE for (tick gui) wasn't given as it is too massive and complicated but the code is commented to an understandable level. Please have a look at the code file to learn more about it.

The **(key-handler)** function is a feature introduced by **(require 2htdp/universe)** library. This function allows responsive output according to the **valid** key input given by the user. "UP" and "SPACE" keys only work when the game is being played and it moves the DORAEMON up by a certain number of pixels. "ENTER" button only works in the STARTING-SCREEN to enter the game-screen. "Q" key works at any time and it is used to stop the game from running any further.

```
(define (key-handler gui KeyEvent)
 (cond
   [(and (or (string=? "up" KeyEvent)
            (string=? " " KeyEvent))
        (boolean=? (GUI-dead (collision-detector gui)) #f))
    (make-GUI (... GUI ...))]
   [(and (string=? KeyEvent "\r") (boolean=? (GUI-dead gui) #f)
        (boolean=? (GUI-game gui) #f))
    (make-GUI (... GUI ...))]
   [(string=? "q" KeyEvent)
    (make-GUI (... GUI ...))]
   [else gui]))
```