Writing Interactive Applications with Base R using `getGraphicsEvent()`

useR! Conference 2026

Warsaw, Poland

July 6-9, 2026

Wolfgang Viechtbauer
Maastricht University

2026-07-07

A Note About These Slides

- the talk is about interactive applications
- during the talk, I did a live demo of several examples
- these static slides cannot reflect the actual contents of the talk

Graphics in R

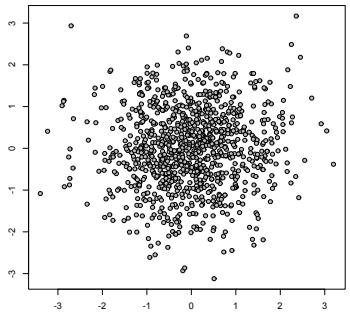
- in most cases seen as **static** / **non-interactive**
- (but: https://cran.r-project.org/view=DynamicVisualizations)
- used to display data, mathematical functions, maps, ...
- interactive graphics / apps are typically built using **external frameworks** (e.g., Shiny[1]) or by interfacing with **other programming languages** (e.g., Tcl/Tk[2], JavaScript)
- `grDevices::getGraphicsEvent()` enables interactive graphics directly within R
- but **rarely used**: only 27 packages on CRAN make use of it

[1] https://shiny.posit.co
https://cran.r-project.org/package=shiny (1084 packages)

[2] https://en.wikipedia.org/wiki/Tcl_(programming_language)
https://search.r-project.org/R/refmans/tcltk/html/00Index.html (116 packages)

Example for `getGraphicsEvent()`

- let's look at the example from `help(getGraphicsEvent)`
- `example01_example_from_getGraphicsEvent.r`



Basic Ideas Behind `getGraphicsEvent()`

- works only on the `windows()` and `x11()` plotting devices
- the basic structure:[1]

```
getGraphicsEvent(
   onMouseDown = NULL, # fun to respond to mouse clicks
   onMouseMove = NULL, # fun to respond to mouse movement
   onMouseUp = NULL,   # fun to respond to mouse button releases
   onKeybd = NULL,     # fun to respond to key presses
   ...)                # a few more arguments
```

- mouse event handlers: `function(buttons, x, y)`
- keyboard event handlers: `function(key)`
- stops when an event handler returns a non-`NULL` value

[1] technically this is the old way of using this functionality; the modern way uses `setGraphicsEventHandlers()` but the idea is the same

`example02_a_very_simple_example.r`

```
plot(mpg ~ wt, data=mtcars, pch=21, bg="gray")

pressed <- FALSE

fun.mousedown <- function(buttons, x, y) {
   print(c(x=x, y=y, button=buttons))
   pressed <<- TRUE
   return(NULL)
}

fun.mousemove <- function(buttons, x, y) {
   if (pressed) print(c(x=x, y=y, button=buttons))
   return(NULL)
}

fun.mouseup <- function(buttons, x, y) {
   pressed <<- FALSE
   return(NULL)
}

fun.keyboard <- function(key) {
   print(key)
   if (key == "q") return(invisible(1))
   return(NULL)
}

getGraphicsEvent(onMouseDown = fun.mousedown,
                 onMouseMove = fun.mousemove,
                 onMouseUp   = fun.mouseup,
                 onKeybd     = fun.keyboard)
```

The `plotannotate` Package

- this led to the `plotannotate` package
- https://cran.r-project.org/package=plotannotate
- interactively annotate graphics with freehand drawing, symbols (points, lines, arrows, rectangles, circles, ellipses), and text
- useful for teaching and creating quick sketches
- `example03_plotannotate.r`

```
library(plotannotate)

plot(mpg ~ wt, data=mtcars, pch=21, bg="gray",
     xlab="Weight", ylab="Miles Per Gallon")

annotate()
```


The Chess Trainer

- I am an avid (but terrible) chess player
- wanted to practice chess openings[1]
- over the 2024 Xmas break, wrote a prototype in R to display a chess board on which one could move the pieces
- can then enter sequences of moves and repeat them as a quiz
- this turned into the `chesstrainer` package
- **repo:** https://github.com/wviechtb/chesstrainer
- **docs:** https://wviechtb.github.io/chesstrainer/

[1] to delay the inevitable moment where my kids start beating me in chess ...

The Chess Trainer

- stateful application architecture
- event-driven graphical user interface
- keyboard and mouse interaction
- multiple application screens/modes
- configurable settings that persist across sessions
- implemented entirely in base R (no Shiny, Tcl/Tk, or some external GUI toolkit)

A Lesson Learned

- functional programming can become awkward for such apps
- functions need lots of state variables, so many calls are of the form: `fun(var1, var2, var3, ..., var100)`
- started to make use of a custom package environment to store state variables, which functions can pull in

Redrawing Images

- the pieces are `png` files drawn with the `png` package
- expected redrawing of pieces on movement to be too slow
- but turns out that I was wrong ...
- `example04_move_piece.r`

The Idle Event Handler

- the idle event handler `onIdle` is a function without arguments
- will be called whenever the event queue is empty
- allows for animations (that could respond to user input)
- only implemented for the `x11()` device
- could be used to write games
- `example05_snake.r`

A Whole New World of (Silly) Possibilities

- writing an R IDE in R itself
- `example06_rstudio_inception.r`

A Whole New World of (Silly) Possibilities

- writing an R IDE in R itself
- `example06_rstudio_inception.r`
- and for one final example ...
- `example07_finish.r`

13

Thank You for Your Attention!

Questions, Comments, Suggestions?

wolfgang.viechtbauer@maastrichtuniversity.nl

https://www.wvbauer.com

@wviechtb @wviechtb