Package 'karel'

August 6, 2021

Title Learning programming with Karel the robot

Version 0.1.0

Description This is the R implementation of Karel the robot, a programming language created by Dr. R. E. Pattis at Stanford University in 1981. Karel is an useful tool to teach introductory concepts about general programming, such as algorithmic decomposition, conditional statements, loops, etc., in an interactive and fun way, by writing programs to make Karel the robot achieve certaing tasks in the world she lives in. Originally based on Pascal, Karel was implemented in many languages through these decades, including 'Java', 'C++', 'Ruby' and 'Python'. This is the first package implementing Karel in R.

Depends R (>= 3.6.0)

Imports purrr, dplyr, tidyr, ggplot2, magrittr, gganimate, gifski

License GPL-2

Encoding UTF-8

Language en, es

RoxygenNote 7.1.1

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

URL https://mpru.github.io/karel/

BugReports https://github.com/mpru/karel/issues/

NeedsCompilation no

Author Marcos Prunello [aut, cre, cph]
(<https://orcid.org/0000-0002-9611-527X>)

Maintainer Marcos Prunello <marcosprunello@gmail.com>

Repository CRAN

Date/Publication 2021-08-06 07:50:02 UTC

R topics documented:

acciones 2
actions 3
cargar_super_karel 4
condiciones 4
conditions 6
ejecutar_acciones 8
generar_mundo 8
generate_world 10
get_pkg_env 11
load_super_karel 12
plot_static_world 12
run_actions 13

Index **14**

`acciones` *Acciones que Karel puede realizar*

Description

`avanzar()`, `girar_izquierda()`, `juntar_coso()` y `poner_coso()` son las cuatro actividades básicas que Karel sabe realizar. Si se habilitan los superpoderes de Karel con `cargar_super_karel()`, entonces también puede `girar_derecha()` y `darse_vuelta()`.

Usage

```
avanzar()

girar_izquierda()

poner_coso()

juntar_coso()

girar_derecha()

darse_vuelta()
```

Value

Estas funciones no devuelven nada, pero realizan cambios en el mundo de Karel que se ven cuando se ejecutan todas las acciones con `ejecutar_acciones()`.

See Also

`cargar_super_karel` `generar_mundo` `ejecutar_acciones`

Examples

```
generar_mundo("mundo001")
avanzar()
juntar_coso()
girar_izquierda()
poner_coso()
ejecutar_acciones()
```

`actions` *Available actions for Karel*

Description

`move()`, `turn_left()`, `pick_beeper()` y `put_beeper()` are the four basic activities that Karel can perform. If you turn on Karel's superpowers with `load_super_karel()`, then she can also `turn_right()` y `turn_around()`.

Usage

```
move()

turn_left()

put_beeper()

pick_beeper()

turn_right()

turn_around()
```

Value

These functions don't return anything, but make changes in Karel's world that are visible when all the actions are run through `run_actions()`.

See Also

`load_super_karel` `generate_world` `run_actions`

Examples

```
generate_world("mundo001")
move()
pick_beeper()
turn_left()
put_beeper()
```

```
run_actions()
```

`cargar_super_karel` *Habilitar los superpoderes de Karel*

Description

Luego de correr `cargar_super_karel()`, Karel también puede girar a la derecha y darse vuelta, a través de las acciones `girar_derecha()` y `darse_vuelta()`. Si no se cargan los superpoderes, estas dos funciones no están disponibles.

Usage

```
cargar_super_karel()
```

Value

No devuelve ningún valor, pero adjuntan al Global Environment las funciones `girar_derecha()` y `darse_vuelta()`.

See Also

`acciones` `generar_mundo` `ejecutar_acciones`

Examples

```
generar_mundo("mundo001")
cargar_super_karel()
darse_vuelta()
girar_derecha()
ejecutar_acciones()
```

`condiciones` *Condiciones que Karel puede verificar*

Description

Este conjunto de funciones devuelven un valor lógico `TRUE` o `FALSE` según la evaluación que Karel puede hacer de su mundo.

Usage

```
frente_abierto()

frente_cerrado()

izquierda_abierto()

izquierda_cerrado()

derecha_abierto()

derecha_cerrado()

hay_cosos()

no_hay_cosos()

karel_tiene_cosos()

karel_no_tiene_cosos()

mira_al_este()

mira_al_norte()

mira_al_oeste()

mira_al_sur()
```

Details

Las funciones `frente_abierto()`, `frente_cerrado()`, `izquierda_abierto()`, `izquierda_cerrado()`, `derecha_abierto()` y `derecha_cerrado()` analizan si hay paredes al frente, a la izquierda o a la derecha de Karel. Las funciones `hay_cosos()` y `no_hay_cosos()` analizan si hay cosos en la posición actual de Karel. Las funciones `karel_tiene_cosos()` y `karel_no_tiene_cosos()` analizan si Karel tiene cosos en su mochila (no visibles en la representación gráfica). Las funciones `mira_al_este()`, `mira_al_oeste()`, `mira_al_norte()` y `mira_al_sur()` analizan la dirección hacia la cual Karel está mirando.

Value

Valor lógico TRUE o FALSE.

See Also

`generar_mundo`

Examples

```
generar_mundo("mundo001")
frente_abierto()
frente_cerrado()
izquierda_abierto()
izquierda_cerrado()
derecha_abierto()
derecha_cerrado()
hay_cosos()
no_hay_cosos()
karel_tiene_cosos()
karel_no_tiene_cosos()
mira_al_este()
mira_al_oeste()
mira_al_norte()
mira_al_sur()
```

---

conditions *Conditions that Karel can test*

Description

These group of functions return a logical value TRUE o FALSE according to Karel's evaluation of her world.

Usage

```
front_is_clear()

front_is_blocked()

left_is_clear()

left_is_blocked()

right_is_clear()

right_is_blocked()

beepers_present()

no_beepers_present()

karel_has_beepers()

karel_has_no_beepers()
```

`facing_east()`

`facing_west()`

`facing_north()`

`facing_south()`

Details

The functions `front_is_clear()`, `front_is_blocked()`, `left_is_clear()`, `left_is_blocked()`, `right_is_clear()` y `right_is_blocked()` test if there is a wall in front of Karel, to her left or to her right, respectively. The functions `beepers_present()` y `no_beepers_present()` test if there are or there are not `beepers` at Karel's current position. The functions `karel_has_beepers()` y `karel_has_no_beepers()` test if Karel has or hasn't got `beepers` in her bag (not visible in the plot). The functions `facing_east()`, `facing_west()`, `facing_north()` y `facing_south()` test the direction to which Karel is facing right now.

Value

Logical value TRUE or FALSE.

See Also

`generate_world`

Examples

```
generate_world("mundo001")
front_is_clear()
front_is_blocked()
left_is_clear()
left_is_blocked()
right_is_clear()
right_is_blocked()
beepers_present()
no_beepers_present()
karel_has_beepers()
karel_has_no_beepers()
facing_east()
facing_west()
facing_north()
facing_south()
```

ejecutar_acciones *Ejecutar acciones*

Description

Esta función produce la animación que muestra todas las acciones realizadas por Karel desde que su mundo fue generado con `generar_mundo`.

Usage

```
ejecutar_acciones(repetir = TRUE)
```

Arguments

`repetir`	Valor lógico TRUE o FALSE que indica si la animación debe repetirse una y otra vez luego de finalizada (por defecto: TRUE).

Value

Produce una animación con `gganimate`.

See Also

`generar_mundo`

Examples

```
generar_mundo("mundo001")
avanzar()
juntar_coso()
girar_izquierda()
poner_coso()
ejecutar_acciones()
```

generar_mundo *Generar el mundo de Karel*

Description

Esta función toma un "mundo" (es decir, una lista con información acerca de su tamaño, paredes, "cosos" presentes y la ubicación y dirección de Karel), lo grafica y prepara todo para que Karel pueda realizar sus acciones. Siempre debe ser evaluada antes de que Karel empiece a cumplir sus objetivos, en especial, si en algún momento hemos cometido un error, debemos comenzar de nuevo corriendo primero esta función.

Usage

```
generar_mundo(mundo)
```

Arguments

mundo	Un caracter de largo 1 indicando el nombre de uno de los mundos que ya vienen en el paquete o un objeto de tipo lista con todos los componentes que debe tener un mundo (ver más abajo en Detalles).

Details

Luego de correr `generar_mundo()`, se ejecutan las acciones de Karel y se pueden visualizar con la función `ejecutar_acciones()`.

El argumento `mundo` puede consistir de un mundo creado (es decir, inventado) por cualquiera. En este caso, `mundo` debe ser una lista con los siguientes componentes:

1. `nx`: TODO
2. `ny`:
3. `hor_walls`:
4. `ver_walls`:
5. `karel_x`:
6. `karel_y`:
7. `karel_dir`:
8. `beepers_x`:
9. `beepers_y`:
10. `beepers_n`:
11. `beepers_bag`:

Value

Dibuja el estado inicial del mundo de Karel y deja todo preparado para comenzar a registrar sus acciones.

See Also

`acciones` `ejecutar_acciones`

Examples

```
generar_mundo("mundo001")
```

`generate_world` *Create Karel's world*

Description

This function takes a "world" (i.e. a list with data about its size, walls, beepers and Karel's position and direction), plots it and prepares everything so that Karel can start performing actions in it. It must be run always before Karel starts working on her goals, especially if we have made a mistake, we must start all over again by first running this function.

Usage

```
generate_world(world)
```

Arguments

`world`	Character vector of length 1 with the name of one of the provided worlds in the package or a list provided by the user with all the components that a world needs (see more below in details).

Details

After running `generate_mundo()`, we can run Karel's actions and finally visualize it all with the function `run_actions()`.

Argument `world` can be create by the user. In this case, it must be a list with the following components:

1. `nx`: TODO
2. `ny`:
3. `hor_walls`:
4. `ver_walls`:
5. `karel_x`:
6. `karel_y`:
7. `karel_dir`:
8. `beepers_x`:
9. `beepers_y`:
10. `beepers_n`:
11. `beepers_bag`:

Value

Plots the initial state of Karel's world and prepares everything to start recording her actions.

See Also

actions run_actions

Examples

```
generate_world("mundo001")
```

---

get_pkg_env *Get Karel's environment*

Description

This function returns the environment called pkg_env created by the package. It's useful for debugging and checking. It's an internal function.

Usage

```
get_pkg_env()
```

Details

pkg_env is an environment created inside the package to store and share between functions all the objects related to Karel's world and its state. Since the functions that will be used by the students should be simple and without arguments (for example, move()), these functions modify internally pkg_env.

The components of this environment are:

1. nx: TODO
2. ny:
3. hor_walls:
4. ver_walls:
5. open_moves:
6. karel:
7. dir_now:
8. x_now:
9. y_now:
10. moment:
11. beepers_any:
12. beepers_bag:
13. beepers_now:
14. beepers_all:
15. base_plot:

Value

An enviroment with objects that represent Karel's world.

`load_super_karel` *Turn on Karel's superpowers*

Description

After running `load_super_karel()`, Karel can also turn right and turn around with `turn_right()` and `turn_around()`. If these superpowers aren't loaded, then these functions won't be available and Karel can't use them.

Usage

```
load_super_karel()
```

Value

It doesn't return anything but attaches to the global environment the functions `turn_right()` and `turn_around()`.

See Also

`actions` `generate_world` `run_actions`

Examples

```
generate_world("mundo001")
load_super_karel()
turn_around()
turn_right()
run_actions()
```

`plot_static_world` *Plot the world at a given time*

Description

This function plots Karel'w wort at the requested time. Initially, time is 1 and with each action that Karel performs, time is incremented by one. Current time is stored in `pkg_env$moment`. This function is useful for debuggint and to get static images to be used in the examples in the handouts for studentes.

Usage

```
plot_static_world(time)
```

Arguments

time	The requested time

Value

Prints the plot.

`run_actions` *Run actions*

Description

This function produces the animation that shows all actions performed by Karel since its world was generated by `generate_world`.

Usage

```
run_actions(loop = TRUE)
```

Arguments

loop	A logical value TRUE or FALSE indicating if the animation should repeat itself after finished or not (defaults to TRUE).

Value

Produces an animation with `gganimate`.

See Also

`generate_world`

Examples

```
generate_world("mundo001")
move()
pick_beeper()
turn_left()
put_beeper()
run_actions()
```

Index

acciones, 2, *4*, *9*
actions, 3, *11*, *12*
avanzar *(*acciones*)*, 2

beepers_present *(*conditions*)*, 6

cargar_super_karel, *2*, 4
condiciones, 4
conditions, 6

darse_vuelta *(*acciones*)*, 2
derecha_abierto *(*condiciones*)*, 4
derecha_cerrado *(*condiciones*)*, 4

ejecutar_acciones, *2*, *4*, 8, *9*

facing_east *(*conditions*)*, 6
facing_north *(*conditions*)*, 6
facing_south *(*conditions*)*, 6
facing_west *(*conditions*)*, 6
frente_abierto *(*condiciones*)*, 4
frente_cerrado *(*condiciones*)*, 4
front_is_blocked *(*conditions*)*, 6
front_is_clear *(*conditions*)*, 6

generar_mundo, *2*, *4*, *5*, *8*, 8
generate_world, *3*, *7*, 10, *12*, *13*
get_pkg_env, 11
girar_derecha *(*acciones*)*, 2
girar_izquierda *(*acciones*)*, 2

hay_cosos *(*condiciones*)*, 4

izquierda_abierto *(*condiciones*)*, 4
izquierda_cerrado *(*condiciones*)*, 4

juntar_coso *(*acciones*)*, 2

karel_has_beepers *(*conditions*)*, 6
karel_has_no_beepers *(*conditions*)*, 6
karel_no_tiene_cosos *(*condiciones*)*, 4
karel_tiene_cosos *(*condiciones*)*, 4

left_is_blocked *(*conditions*)*, 6
left_is_clear *(*conditions*)*, 6
load_super_karel, *3*, 12

mira_al_este *(*condiciones*)*, 4
mira_al_norte *(*condiciones*)*, 4
mira_al_oeste *(*condiciones*)*, 4
mira_al_sur *(*condiciones*)*, 4
move *(*actions*)*, 3

no_beepers_present *(*conditions*)*, 6
no_hay_cosos *(*condiciones*)*, 4

pick_beeper *(*actions*)*, 3
plot_static_world, 12
poner_coso *(*acciones*)*, 2
put_beeper *(*actions*)*, 3

right_is_blocked *(*conditions*)*, 6
right_is_clear *(*conditions*)*, 6
run_actions, *3*, *11*, *12*, 13

turn_around *(*actions*)*, 3
turn_left *(*actions*)*, 3
turn_right *(*actions*)*, 3