Package: karel (via r-universe)

August 26, 2024

Title Learning programming with Karel the robot

Version 0.1.1.9001

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 certain 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, cli

License GPL-2

Encoding UTF-8

LazyData true

Language en, es

RoxygenNote 7.3.1

Suggests testthat (>= 3.0.0), knitr, rmarkdown, vdiffr

VignetteBuilder knitr

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

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

Config/testthat/edition 3

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/karel

RemoteRef master

RemoteSha 439aa6d0ff46aeb13979ead83cc903acdd59db3f

Contents

`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()
if (interactive()) 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()
if (interactive()) 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()
if (interactive()) 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_oeste()

mira_al_norte()

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 or 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()` and `no_beepers_present()` test whether there are any `beepers` at Karel's current position. The functions `karel_has_beepers()` and `karel_has_no_beepers()` test if Karel has any `beepers` in her bag (not visible in the plot). The functions `facing_east()`, `facing_west()`, `facing_north()` and `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()
```

`conseguir_amb` *Obtener el ambiente de Karel*

Description

Esta función devuelve un ambiente (R environment) llamado pkg_env, que es creado por el paquete. Se puede usar para probar el funcionamiento del paquete. Es una función interna, no está pensada para ser usada por estudiantes, pero se puede usar con karel:::conseguir_amb().

Usage

```
conseguir_amb()
```

Details

`pkg_env` es un ambiente de R creado dentro del paquete para guardar y compartir entre las funciones todos los objetos relacionados con el mundo de Karel y su estado en cada momento. Dado que estas funciones que usan los estudiantes deben ser simples y no usar argumentos (como, por ejemplo, `avanzar()`) estas funciones modifican internamente a `pkg_env` para implementar cada acción.

Los componentes de este ambiente son:

1. `nx`: tamaño del mundo de Karel, número de celdas en el eje x
2. `ny`: tamaño del mundo de Karel, número de celdas en el eje x
3. `hor_walls`: un data.frame con una fila por cada pared horizontal que hay en el mundo de Karel y 3 columnas: x (coordenada del inicio de la pared en el eje x), y (coordenada del inicio de la pared en el eje y), lgth (longitud de la pared, en cantidad de celdas que abarca). Si toma el valor NULL, no hay paredes horizontales en el mundo.
4. `ver_walls`: un data.frame con una fila por cada pared vertical que hay en el mundo de Karel y 3 columnas: x (coordenada del inicio de la pared en el eje x), y (coordenada del inicio de la pared en el eje y), lgth (longitud de la pared, en cantidad de celdas que abarca). Si toma el valor NULL, no hay paredes verticales en el mundo.
5. `open_moves`: un arreglo de dimensión nx x ny x 4 de valores TRUE/FALSE que indica si Karel puede moverse en cada dirección desde una posición determinada. Para ejemplo, si Karel está en la esquina inferior izquierda, que es la celda [1, 1], no puede ir al sur ni a la izquierda, por lo que tenemos open_moves[1, 1, 3] y open_moves[1, 1, 4] establecido en FALSO. Dependiendo de las paredes existentes podría moverse al sur o al norte, por lo que open_moves[1, 1, 1] y open_moves[1, 1, 2] puede ser VERDADERO o FALSO. Teniendo en cuenta el tamaño del mundo y la paredes, este arreglo es creado por la función interna `generate_open_moves`.
6. `karel`: un data.frame con una fila para cada momento, en el que se registra cada estado de Karel a lo largo de la ejecución de sus acciones. Tiene 4 columnas: karel_x (coordenada de Karel en el eje x), karel_y (coordenada de Karel en el eje y), karel_dir (dirección a la que mira Karel, 1 este, 2 norte, 3 oeste o 4 sur), y moment (valor entero indicando cada momento).
7. `dir_now`: dirección en la cual Karel está mirando ahora.

8. x_now: coordenada en el eje x actual de Karel.
9. y_now: coordenada en el eje y actual de Karel.
10. moment: momento actual (valor entero).
11. beepers_any: cantidad total de cosos presentes en el mundo en este momento.
12. beepers_bag: número de cosos que Karel tienen a disposición en su muchila ahora. Karel puede poner cosos si es que tiene cosos en su mochila. Puede tomar el valor Inf.
13. beepers_now: un data.frame con tantas filas como celdas con cosos haya en el mundo y 5 columnas: x y y para las coordenadas de la celda, cell es el número de la celda contando como celda número 1 la celda en la esquina inferior izquierda y yendo hacia arriba por fila (lo que significa que la celda número 2 sería la celda en las coordenadas x=2 e y=1), n el número de cosos en esta celda y moment el momento al cual corresponde este estado del mundo. Es creado por la función interna create_beepers.
14. beepers_all: un data.frame con la misma estructura que beepers_now. Mientras que beepers_now solo tiene el estado actual de cosos, beepers_all acumula todos los estados para la animación, uniendo las filas de beepers_now y beepers_all después de cada acción.
15. base_plot: gráfico inicial del mundo, con su tamaño y todas las paredes si las hay. No muestra a Karel ni a los cosos, ya que estos pueden cambiar con el tiempo. Este es el gráfico base que es utilizado más tarde para producir la animación. Este gráfico es creado por la función interna plot_base_world.

Value

Un ambiente de R con objetos que representan al mundo de Karel.

Examples

```
generar_mundo("mundo001")
if (interactive()) karel:::conseguir_amb()
```

---

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 = FALSE)
```

Arguments

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

Value

Produce una animación con `gganimate`.

See Also

`generar_mundo`

Examples

```
generar_mundo("mundo001")
avanzar()
juntar_coso()
girar_izquierda()
poner_coso()
if (interactive()) 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 carácter 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`: tamaño del mundo de Karel, número de celdas en el eje x
2. `ny`: tamaño del mundo de Karel, número de celdas en el eje x

3. `hor_walls`: un data.frame con una fila por cada pared horizontal que hay en el mundo de Karel y 3 columnas: x (coordenada del inicio de la pared en el eje x), y (coordenada del inicio de la pared en el eje y), lgth (longitud de la pared, en cantidad de celdas que abarca). Si toma el valor NULL, no hay paredes horizontales en el mundo.
4. `ver_walls`: un data.frame con una fila por cada pared vertical que hay en el mundo de Karel y 3 columnas: x (coordenada del inicio de la pared en el eje x), y (coordenada del inicio de la pared en el eje y), lgth (longitud de la pared, en cantidad de celdas que abarca). Si toma el valor NULL, no hay paredes verticales en el mundo.
5. `karel_x`: coordenada en el eje x para la posición inicial de Karel.
6. `karel_y`: coordenada en el eje y para la posición inicial de Karel.
7. `karel_dir`: dirección inicial de Karel: 1 (mira al oeste), 2 (mira al norte), 3 (mira al oeste) o 4 (mira al sur).
8. `beepers_x`: vector numérico con las coordenadas en el eje x de las celdas donde hay cosos inicialmente. El largo de los vectores beepers_x, beepers_y y beepers_n debe coincidir. Si no se desea que haya cosos en el mundo, proveer el valor NULL.
9. `beepers_y`: vector numérico con las coordenadas en el eje y de las celdas donde hay cosos inicialmente. El largo de los vectores beepers_x, beepers_y y beepers_n debe coincidir. Si no se desea que haya cosos en el mundo, proveer el valor NULL.
10. `beepers_n`: vector numérico con la cantidad de cosos que hay inicialmente en cada una de las posiciones determinadas por los valores de beepers_x y beepers_y. El largo de los vectores beepers_x, beepers_y y beepers_n debe coincidir. Si no se desea que haya cosos en el mundo, proveer el valor NULL.
11. `beepers_bag`: número de cosos que Karel tienen a disposición en su muchila al inicio. Karel puede poner cosos si es que tiene cosos en su mochila. Puede tomar el valor Inf.

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_world()`, 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`: size of Karel's world, number of cells in x-axis.
2. `ny`: size of Karel's world, number of cells in y-axis.
3. `hor_walls`: a data.frame with a row for each horizontal wall in Karel's world and 3 columns: x (coordinate of the start of the wall in the x axis), y (coordinate of the start of the wall in the y axis), lgth (length of the wall, in number of cells it covers). If it is NULL, there are no horizontal walls in the world.
4. `ver_walls`: a data.frame with a row for each vertical wall in Karel's world and 3 columns: x (coordinate of the start of the wall in the x axis), y (coordinate of the start of the wall in the y axis), lgth (length of the wall, in number of cells it covers). If it takes the value NULL, there are no vertical walls in the world.
5. `karel_x`: x-coordinate for Karel's initial position.
6. `karel_y`: y-coordinate for Karel's initial position.
7. `karel_dir`: Karel's starting direction: 1 (facing west), 2 (facing north), 3 (facing west), or 4 (facing south).
8. `beepers_x`: Numeric vector with the x-axis coordinates of the cells where there are beepers initially. The length of the vectors beepers_x, beepers_y and beepers_n must match. If you don't want beepers in the world, supply the value NULL.

9. beepers_y: Numeric vector with the coordinates in the y-axis of the cells where there are beepers initially. The length of the vectors beepers_x, beepers_y and beepers_n must match. If you don't want beepers in the world, supply the value NULL.
10. beepers_n: numeric vector with the number of beepers that are initially in each of the positions determined by the values of beepers_x and beepers_y. The length of the vectors beepers_x, beepers_y and beepers_n must match. If you don't want beepers in the world, supply the value NULL.
11. beepers_bag: number of beepers that Karel has available in its bag at the beginning. Karel can put beepers if it has beepers in its bag. It can take the value Inf.

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, not thought to be used by students, but can be used with karel:::get_pkg_env().

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: size of Karel's world, number of cells in x-axis.
2. ny: size of Karel's world, number of cells in y-axis.

3. `hor_walls`: a data.frame with a row for each horizontal wall in Karel's world and 3 columns: x (coordinate of the start of the wall in the x axis), y (coordinate of the start of the wall in the y axis), lgth (length of the wall, in number of cells it covers). If it is NULL, there are no horizontal walls in the world.
4. `ver_walls`: a data.frame with a row for each vertical wall in Karel's world and 3 columns: x (coordinate of the start of the wall in the x axis), y (coordinate of the start of the wall in the y axis), lgth (length of the wall, in number of cells it covers). If it takes the value NULL, there are no vertical walls in the world.
5. `open_moves`: a nx x ny x 4 array of TRUE/FALSE values indicating if Karel can move to each direction from a given position. For example, if Karel is in the bottom left corner, which is cell [1, 1], it can't go south or left, so we have both open_moves[1, 1, 3] and open_moves[1, 1, 4] set to FALSE. Depending on the existing walls it could move south or north, so open_moves[1, 1, 1] and open_moves[1, 1, 2] could be TRUE or FALSE. Taking into account the size of the world and the walls, this array is created by the internal function `generate_open_moves`.
6. `karel`: a data.frame with a row for each moment, in which each state of Karel is recorded throughout the execution of its actions. It has 4 columns: karel_x (Karel's x-axis coordinate), karel_y (Karel's y-axis coordinate), karel_dir (the direction Karel is facing, 1 east, 2 north, 3 west, or 4 south), and moment (integer value indicating each moment).
7. `dir_now`: current Karel's facing direction.
8. `x_now`: x-axis coordinate of Karel's current position.
9. `y_now`: y-axis coordinate of Karel's current position.
10. `moment`: current moment (integer value).
11. `beepers_any`: total amount of beepers present in the world at this moment.
12. `beepers_bag`: number of beepers that Karel has available in its bag at the moment. Karel can put beepers if it has beepers in its bag. It can take the value Inf.
13. `beepers_now`: a data.frame with as many rows as cells with beepers in the world and 5 columns: `x` and `y` for the coordinates of the cell, `cell` is the number of the cell counting as cell number 1 the cell in the bottom left corner and going upwards by row (meaning cell number 2 would be the cell in coordinates x=2 and y=1), `n` the number of beepers in this cell and `moment` the moment in which this state of the world corresponds to. It is created by the internal function `create_beepers`.
14. `beepers_all`: a data.frame with the same structure as `beepers_now`. While `beepers_now` only has current state of beepers, `beepers_all` acummulates all states for the animation, binding the rows of `beepers_now` and `beepers_all` after each action.
15. `base_plot`: the initial plot of the world, with its size and all the walls if there are any. It doesn't show Karel or the beepers, since those things can change with time. This is the base plot that is used later to produce the animation. This plot is created by the internal function `plot_base_world`.

Value

An enviroment with objects that represent Karel's world.

Examples

```
generate_world("mundo001")
if (interactive()) karel:::get_pkg_env()
```

`graficar_mundo_estatico` *Producir un gráfico del mundo de Karel en un momento dado*

Description

Esta función grafica el mundo de Karel en el momento pedido. Inicialmente, momento toma el valor 1 y con cada acción que Karel realiza se incrementa en 1. El momento actual está guardado en `pkg_env$moment`. Esta función es útil para revisar el código y para obtener imágenes estáticas que pueden usarse al crear ejemplos y ejercicios en los materiales de estudio para los estudiantes. Es una función interna, no está pensada para ser usada por estudiantes, pero se puede usar con karel:::graficar_mundo_estatico().

Usage

```
graficar_mundo_estatico(momento)
```

Arguments

`momento`	El momento que se desea graficar.

Value

Imprime el gráfico.

Examples

```
if (interactive()) karel:::graficar_mundo_estatico(1)
```

`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()
if (interactive()) 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 debugging and to get static images to be used in the examples in the handouts for students. It's an internal function, not thought to be used by students, but can be used with karel:::plot_static_world().

Usage

```
plot_static_world(time)
```

Arguments

`time`	The requested time

Value

Prints the plot.

Examples

```
if (interactive()) karel:::plot_static_world(1)
```

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 = FALSE)
```

Arguments

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

Value

Produces an animation with `gganimate`.

See Also

`generate_world`

Examples

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

Index