Package ‘networkGen’

July 22, 2025

Type Package

Title Network Maze Generator

Version 0.1.1

Date 2017-12-04

Maintainer Bao Sheng Loe (Aiden) <bsl28@cam.ac.uk>

Description A network Maze generator that creates different types of network mazes.

License GPL-3

Imports igraph, mgcv, stats

LazyData TRUE

RoxygenNote 6.0.1

NeedsCompilation no

Author Bao Sheng Loe (Aiden) [aut, cre, cph]

Repository CRAN

Date/Publication 2017-12-04 17:31:37 UTC

Contents

`check.graph` *check.graph*

Description

Simple check function

Usage

```
check.graph(x)
```

Arguments

`x`	This check graphs to confirm that it is a closed loop logic

Details

To ensure that it is a closed loop logic

Author(s)

Aiden Loe

Examples

```
check.graph(nodeLogic(value = 1, type= "circuit", itemFamily= 1))
```

`logicMap` *Logic Map*

Description

This function is used to change the node display.

Usage

```
logicMap(x, base.colour, start.colour, end.colour, names = NULL, newValue,
  default.colour = TRUE, no.label = FALSE)
```

Arguments

`x`	This is taken from the logic derived using the igraph package.
`base.colour`	This is the colour of all the nodes if no colour is specified
`start.colour`	This is the colour of the first node
`end.colour`	This is the colour of the last node
`names`	If names=NUll then use default names in the package
`newValue`	This is the value of the number of nodes from the logic
`default.colour`	If TRUE, then the colours of the node will not change. If FALSE, the colours of the node will change.
`no.label`	If no.labels is TRUE, then it will not print the names in the nodes. If FALSE, then it will be numeric values.

Details

This functions is embedded with `check.graph`, `edge.v`, `colour_display`. For example, you can use this function to add in names inside the node, or change the width of the edge, or to include the labels in the nodes given by the names arg. Generally, it is used for the assisting in the design of the network maze

This allow us to create a map with close looped form

Value

The map item based on the logic of the igraph package

Author(s)

Aiden Loe

Examples

```
logic <- nodeLogic(value = 8, type= "circuit", itemFamily= 1)
names <- c('a','b','c','d','e','f','g')
logicMap(logic, no.label=FALSE, names=names)
```

`netHTML` *Generate Network Maze (No arrows)*

Description

This function generates an network Maze with at most 2 arrows.

Usage

```
netHTML(nodeLogic = NULL, wd = NULL, names = NULL, concerto = "C5")
```

Arguments

`nodeLogic`	This is the connections between the nodes.
`wd`	This is the working directory to save the HTML source code in. If not given, the file will be saved in the default working directory.
`names`	This allows you to put in your own names in the nodes when generating the maze.
`concerto`	Choose between concerto 4 or concerto 5. CSS scale on concerto 5 is slightly off. So if you are not using concerto, you might want to change the default option to concerto 4 instead.

Details

This function creates a maze and is saved into your working directory. At most up to 2 arrows per maze is generated.

Author(s)

Aiden Loe

Examples

```
#create node logic
logic <- nodeLogic(value = 8, type= "circuit", itemFamily= 1)

#Folder to save html/
#setwd("~/desktop")
#filePath<- getwd()

#Generate item
set.seed(1)
netHTML(logic, wd=NULL, names=NULL, concerto="C5")
```

`netHTML1arrow` *Generate Network Maze (1 arrow)*

Description

This function generates an network Maze with 1 arrow.

Usage

```
netHTML1arrow(nodeLogic = NULL, wd = NULL, names = NULL,
  concerto = "C5")
```

Arguments

`nodeLogic`	This is the connections between the nodes.
`wd`	is the working directory to save the HTML source code in. If not given, the file will be saved in the default working directory.
`names`	This allows you to put in your own names in the nodes when generating the maze.
`concerto`	Choose between concerto 4 or concerto 5. So if you are not using concerto, you might want to change the default option to concerto 4 instead.

Details

This function creates a maze and is saved into your working directory. This is regardless of whether it is a trail or circuit type maze. 1 arrow per maze is generated.

Author(s)

Aiden Loe

Examples

```
#create random names
countries <- c("Croatia","Cyprus","Denmark","Finland","France","Germany",
"Greece","Hungary","Iceland","UK","US")

#create node logic
logic <- nodeLogic(value = 8, type= "circuit", itemFamily= 1)

#Folder to save html/
#setwd("~/desktop")
#filePath<- getwd()

#Generate item
set.seed(1)
netHTML1arrow(logic, wd=NULL, names = countries,concerto="C5")
```

`netHTML2arrows` *Generate Network Maze (2 arrows)*

Description

This function generates an network Maze with 2 arrows.

Usage

```
netHTML2arrows(nodeLogic = NULL, wd = NULL, names = NULL,
  concerto = "C5")
```

Arguments

`nodeLogic`	This is the connections between the nodes.
`wd`	is the working directory to save the HTML source code in. If not given, the file will be saved in the default working directory.
`names`	This allows you to put in your own names in the nodes when generating the maze.
`concerto`	Choose between concerto 4 or concerto 5. So if you are not using concerto, you might want to change the default option to concerto 4 instead.

Details

This function creates a maze and is saved into your working directory. This is regardless of whether it is a trail or circuit type maze. 2 arrows per maze is generated.

Author(s)

Aiden Loe

Examples

```
#create random names
countries <- c("Croatia","Cyprus","Denmark","Finland","France","Germany",
"Greece","Hungary","Iceland","UK","US")

#create node logic
logic <- nodeLogic(value = 8, type= "circuit", itemFamily= 1)

#Folder to save html/
#setwd("~/desktop")
#filePath<- getwd()

#Generate item
set.seed(1)
netHTML2arrows(logic, wd=NULL, names = countries,concerto="C5")
```

`netHTML3arrows` *Generate Network Maze (3 arrows)*

Description

This function generates an network Maze with 3 arrows.

Usage

```
netHTML3arrows(nodeLogic = NULL, wd = NULL, names = NULL,
  concerto = "C5")
```

Arguments

`nodeLogic`	This is the connections between the nodes.
`wd`	is the working directory to save the HTML source code in. If not given, the file will be saved in the default working directory.
`names`	This allows you to put in your own names in the nodes when generating the maze.
`concerto`	Choose between concerto 4 or concerto 5. So if you are not using concerto, you might want to change the default option to concerto 4 instead.

Details

This function creates a maze and is saved into your working directory. This is regardless of whether it is a trail or circuit type maze. 3 arrows per maze is generated. Bearing in mind that with 3 arrows, the maze may not always be solved. Hence, it still requires checking prior to using it as a test.

Author(s)

Aiden Loe

Examples

```
#create random names
countries <- c("Croatia","Cyprus","Denmark","Finland","France","Germany",
"Greece","Hungary","Iceland","UK","US")

#create node logic
logic <- nodeLogic(value = 8, type= "circuit", itemFamily= 1)

#Folder to save html/
#setwd("~/desktop")
#filePath<- getwd()

#Generate item
set.seed(1)
netHTML3arrows(logic, wd=NULL, names = countries,concerto="C5")
```

`networkGen` *networkGen: A package for generating network type maze*

Description

The mazeGen package provides functions to generate the Network Mazes.

The item families are not designed to be increasingly difficulty. They are just different at this stage. This package is still at its early stages.

Nevertheless, there are 4 functions in which will help you to start generating your own html network mazes.

- netHTML
- netHTML1arrow
- netHTML2arrows
- netHTML3arrows

References

coming soon

`nodeLogic` *Node Logic*

Description

This function generates the node logic for circuit (9 item family) or trail (6 item family) type maze. Please refer to details for more information

Usage

```
nodeLogic(value, type, itemFamily)
```

Arguments

`value`	seed value
`type`	select either 'circuit' or 'trail' type network maze.
`itemFamily`	There are 9 item family for circuit and 6 item family for trail type network maze.

Details

Currently, there are 9 item families for circuit type items and 6 item families for trail type mazes. They are by no means based on increasing difficulty. This is based on the uniquness of each pattern.

Circuit (radical 1). 2 Same even number nodes

- Item Family 1: In total 4 moves
- Item Family 2: In total 8 moves
- Item Family 3: In total 12 Moves
- Item Family 4: In total 16 of moves

Circuit (radical 2). Different even number nodes

- Item Family 5: In total 6 moves. 1 node with 4 edges (Sample Item 2).
- Item Family 6: In total 9 moves. 2 nodes with 4 edges

- Item Family 7: In total 12 moves. 3 nodes with 4 edges
- Item Family 8: In total 15 moves. 1 node with 6 edges, 2 nodes with 4 edges, the rest with 2 edges
- Item Family 9: In total 12 moves. 1 node with 6 edges, 1 node with 4 edges, the rest with 2 edges.

Trail. Same uneven number of nodes

- Item Family 1: In total 6 moves
- Item Family 2: In total 10 moves
- Item Family 3: In total 14 moves
- Item Family 4: In total 9 moves
- Item Family 5: In total 13 moves
- Item Family 6: In total 10 move

Author(s)

Aiden Loe

Examples

```
nodeLogic(value = 1, type= "circuit", itemFamily= 1)
```

Index