The maze package*

Sicheng Du†

January 10, 2023 v1.2

1 Changes in this version

Thanks to the TEXnicians who kindly gave valuable and earnest suggestions to this package, the version 1.2 is now released. The main changes include

1. Corrected some mistakes in this manual;
2. Modified the format in the source code to improve compatibility;
3. Improved the output map of the maze to make it clearer.

2 User instructions

The maze package can generate random square mazes of a specified size. You need to start from the bottom-left corner and reach the top-right corner to play it.

`\maze` `{`⟨*size*⟩`}[`⟨*seed*⟩`]` is the syntax of the command that generates a maze. Thereinto

`{`⟨***size***⟩`}` controls the density of the walls inside the maze and directly influences its complexity. It must be a positive integer in the range **[2,100]**.

To have the package produce a satisfactory output, it is recommended to input a number between 20 and 50 into `{`⟨*text*⟩`}`. Over large numbers may cause TEX to exhaust its capacity and fail to produce anything.

`[`⟨***seed***⟩`]` is an optional parameter that specifies the seed for random numbers. If it is omitted, the current time (minute) will be used as the seed instead.

As an example, the mazes in Figure 1 can be created by `\maze{30}[4]`, `\maze{20}[7]` and `\maze{25}`[1] respectively.

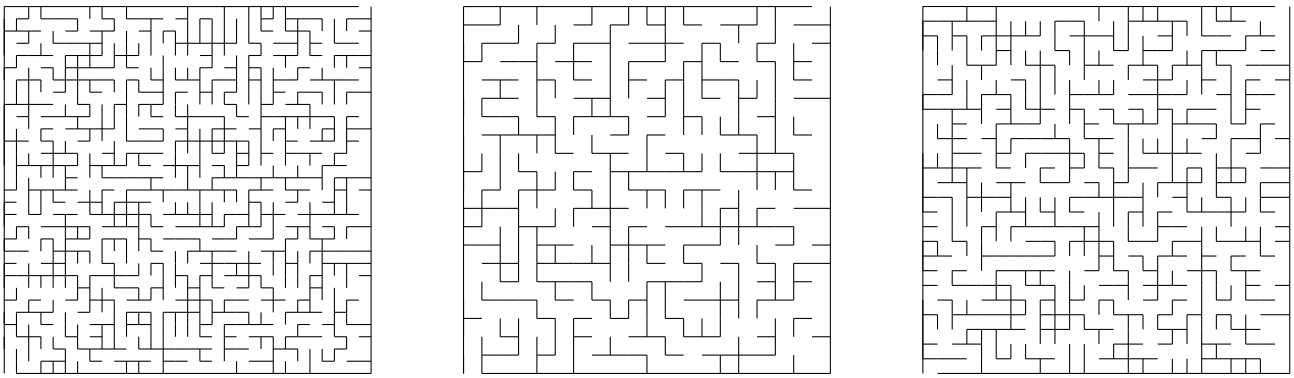

Figure 1: Examples of mazes

* This project is distributed under the LATEX Project Public License, version 1.3c.

† ✉ siddsc@foxmail.com

[1] This maze is likely to look different because of the difference of compiling time.

3 Algorithm and code implementation

This package uses the expl3 programming layer. Under the scope of `\ExplSyntaxOn` we first define some variables

```
\int_new:N\l_maze_rand_int                              % the random variable
\int_new:N\l_maze_old_int \int_new:N\l_maze_new_int
\dim_const:Nn\g_maze_size_dim{\linewidth}               % store the line width
\intarray_new:Nn\g_maze_map_intarray{10000}             % map of the maze
\intarray_new:Nn\g_walls_v_intarray{9900}               % existence of vertical
\intarray_new:Nn\g_walls_h_intarray{9900}               % /horizontal walls
```

The internal command is defined as `\m@ze`. Starting with variable initialization,

```
\newcommand{\m@ze}[2]{
  \sys_gset_rand_seed:n{#2}
  \intarray_gzero:N\g_maze_map_intarray
  \intarray_gzero:N\g_walls_v_intarray
  \intarray_gzero:N\g_walls_h_intarray
  \int_step_inline:nn{#1*#1}{
    \intarray_gset:Nnn\g_maze_map_intarray{##1}{##1}
  }
  \int_step_inline:nn{#1*(#1-1)}{
    \intarray_gset:Nnn\g_walls_v_intarray{##1}{1}
    \intarray_gset:Nnn\g_walls_h_intarray{##1}{1}
  }
```

Then we apply the *Kruskal* algorithm to the intarray-based variant of the *Union-find set*. Our loop should end when the beginning and finishing cells are connected. (so that a path exists)

```
  \bool_do_until:nn{
    \int_compare_p:nNn{\intarray_item:Nn\g_maze_map_intarray{1}}
    ={\intarray_item:Nn\g_maze_map_intarray{#1*#1}}
  }{
    \int_set:Nn\l_maze_rand_int{\int_rand:n{#1*(#1-1)}}
    \int_compare:nNnTF{0}={\intarray_item:Nn\g_walls_v_intarray{\l_maze_rand_int}}{}{
      \int_compare:nNnTF{
        \intarray_item:Nn\g_maze_map_intarray{
          \l_maze_rand_int+\int_div_truncate:nn{\l_maze_rand_int-1}{#1-1}
        }
      }={
        \intarray_item:Nn\g_maze_map_intarray{
          1+\l_maze_rand_int+\int_div_truncate:nn{\l_maze_rand_int-1}{#1-1}
        }
        }{}{
        \int_set:Nn\l_maze_new_int{
          \intarray_item:Nn\g_maze_map_intarray{
            1+\l_maze_rand_int+\int_div_truncate:nn{\l_maze_rand_int-1}{#1-1}
          }
        }
        \int_set:Nn\l_maze_old_int{
          \intarray_item:Nn\g_maze_map_intarray{
            \l_maze_rand_int+\int_div_truncate:nn{\l_maze_rand_int-1}{#1-1}
          }
        }
        \intarray_gset:Nnn\g_walls_v_intarray{\l_maze_rand_int}{0}
        \int_step_inline:nn{#1*#1}{
          \int_compare:nNnTF{\l_maze_old_int}={\intarray_item:Nn\g_maze_map_intarray{##1}}
          {\intarray_gset:Nnn\g_maze_map_intarray{##1}{\l_maze_new_int}}{}
        }
      }
    }
    \int_set:Nn\l_maze_rand_int{\int_rand:n{#1*(#1-1)}}
    \int_compare:nNnTF{0}={\intarray_item:Nn\g_walls_h_intarray{\l_maze_rand_int}}{}{
      \int_compare:nNnTF{\intarray_item:Nn\g_maze_map_intarray{\l_maze_rand_int}}
```

```
      ={\intarray_item:Nn\g_maze_map_intarray{#1+\l_maze_rand_int}}{}{
        \int_set:Nn\l_maze_new_int{
          \intarray_item:Nn\g_maze_map_intarray{#1+\l_maze_rand_int}
        }
        \int_set:Nn\l_maze_old_int{
          \intarray_item:Nn\g_maze_map_intarray{\l_maze_rand_int}
        }
        \intarray_gset:Nnn\g_walls_h_intarray{\l_maze_rand_int}{0}
        \int_step_inline:nn{#1*#1}{
          \int_compare:nNnTF{\l_maze_old_int}={\intarray_item:Nn\g_maze_map_intarray{##1}}
          {\intarray_gset:Nnn\g_maze_map_intarray{##1}{\l_maze_new_int}}{}
        }
      }
    }
  }
}
```

Lastly, we finish off by defining the user command, which outputs a map of the maze. To set the size and draw the boundaries,

```
\NewDocumentCommand\maze{mO{\c_sys_minute_int}}{
  \m@ze{#1}{#2}
  \setlength{\unitlength}{\fp_eval:n{.4/#1}\g_maze_size_dim}
  \begin{picture}(#1,#1)(0,0)
  \put(0,0){\line(0,1){#1}} \put(#1,#1){\line(0,-1){#1}}
  \put(\int_eval:n{#1-1},#1){\line(-1,0){\int_eval:n{#1-1}}}
  \put(1,0){\line(1,0){\int_eval:n{#1-1}}}
```

We extract from `\g_walls_h_intarray` and `\g_walls_v_intarray` and draw a line wherever a wall exists.

```
  \int_step_inline:nn{#1*(#1-1)}{
    \int_compare:nNnTF{0}={\intarray_item:Nn\g_walls_h_intarray{##1}}{}{
      \put(
        \int_mod:nn{##1-1}{#1},
        \int_eval:n{1+\int_div_truncate:nn{##1-1}{#1}}
      ){\line(1,0){1}}
    }
    \int_compare:nNnTF{0}={\intarray_item:Nn\g_walls_v_intarray{##1}}{}{
      \put(
        \int_mod:nn{##1}{#1-1},
        \int_div_truncate:nn{##1-1}{#1-1}
      ){\line(0,1){1}}
    }
  }
  \end{picture}
}
\ExplSyntaxOff
% End of package code
```