

# TileDominoes Group

John D. Baker

<https://github.com/bakerjd99/jackshacks/blob/main/TileDominoes.ijs>

SHA-256: 94ab2f37de348cfd73805bcdaba5ca8f390257a2939d4d30e58effb71aeb6d56

May 21, 2024

## Contents

<b>TileDominoes Overview</b>	<b>2</b>
TileDominoes Interface . . . . .	2
Using TileDominoes . . . . .	2
<b>TileDominoes Source Code</b>	<b>4</b>
<b>=: Index</b>	<b>15</b>

## TileDominoes Overview

**TileDominoes** is a J script that tiles 4x4 square grids with two cell *dominoes*. The original problem comes from *Proofs: a Long-Form Mathematics Textbook* by Jay Cummings. The problem is stated on page seven of my copy.

The J solutions worked out here do more than show the original problem has no solution. It finds all tilings of two cell dominoes (even when *diagonal*) dominoes are allowed for 4x4 grids.

## TileDominoes Interface

```
tiledominoes0 [8] place 2x1 domino tiles on a 4x4 grid
tiledominoes1 [10] 4x4 grid missing first/last cell cannot be tiled by 2x1 vh dominoes
tiledominoes2 [11] count domino tilings of 4x4 grid missing (y) cells
tiledominoes3 [12] list 4x4 tile solutions
tiledominoes4 [13] list 4x4 tile solutions allowing diagonal tiles
```

## Using TileDominoes

The code is self explanatory. See the embedded comments. Here's some typical calls.

```
NB. install jackshacks scripts
install 'github:bakerjd99/jackshacks'
```

```
NB. load TileDominoes script assuming standard install
load '~addons/jacks/TileDominoes.ijs'
```

*NB. numbered grid cells*

`i. 4 4`

*NB. number of solutions to original problem*

*NB. corner cells 0 and 15 are removed - result is 0*

`#tiledominoes3 0 15`

*NB. number of solutions when no cells are removed*

`#tiledominoes3 i.0`

*NB. if diagonal dominoes are added the*

*NB. original problem has many solutions.*

`tiledominoes4 0 15`

## TileDominoes Source Code

```
NB.*TileDominoes s-- words created solving `[_Cummings:2021aa_]` page 7.
NB.
NB. verbatim: interface word(s):
NB. -----
NB. tiledominoes0 - place 2x1 domino tiles on a 4x4 grid
NB. tiledominoes1 - 4x4 grid missing first/last cell cannot be tiled by 2x1 vh dominoes
NB. tiledominoes2 - count domino tilings of 4x4 grid missing (y) cells
NB. tiledominoes3 - list 4x4 tile solutions
NB. tiledominoes4 - list 4x4 tile solutions allowing diagonal tiles
NB.
NB. created: 2024May20
NB. changes: -----
NB. (tiledominoes4) added to explore diagonal tiles

coclass 'TileDominoes'

NB.*end-header

NB. interface words (IFACEWORDSTileDominoes) group
IFACEWORDSTileDominoes=: <;._1 ' tiledominoes0 tiledominoes1 tiledominoes2 tiledominoes3 tiledominoes4'

NB. root words (ROOTWORDSTileDominoes) group
ROOTWORDSTileDominoes=: <;._1 ' IFACEWORDSTileDominoes ROOTWORDSTileDominoes VMDTileDominoes smoutput tiled
>..>ominoes0 tiledominoes1 tiledominoes2 tiledominoes3 tiledominoes4 tilefreq'
```

*NB. version, make count, and date*

VMDTileDominoes=: '0.8.0';11;'21 May 2024 16:15:43'

*NB. signal with optional message*

assert=: 0 0" \_ \$ 13!:8^:((0: e. ])^(12"\_))

comb=: 4 : 0

*NB.\*comb v-- all size (x) combinations of i.y*

*NB.*

*NB. dyad: it =. iaR comb iaN*

*NB.*

*NB. 3 comb 5 NB. 5 chose 3 combinations*

*NB. (i. >:5) comb&.> 5 NB. note empty and complete*

k=. i.>:d=.y-x

z=. (d\$<i.0 0),<i.1 0

for. i.x do. z=. k ,.&.> ,&.>/\ . >:&.> z end.

; z

)

findtilings=: 4 : 0

*NB.\*findtilings v-- finds tilings by testing all possible slot combinations.*

*NB.*

*NB. dyad: btilSlots =. itSlots findtilings ilExcludeCells*

```
NB. only even numbers of cells may be excluded
NB. as any tiling consists of an even number of cells
'cell number(s) invalid' assert (y e. ,x) *. 0=2|#y

NB. exclude (y) cells
slots= . x #~ -. +./"1 x e. y

NB. cells to be covered
cells= . ~. ,slots

NB. generate all possible tile slot combinations
tilings= . , "2 ((-:#cells) comb #slots) { slots

NB. a solution must cover all cells and
NB. no cell must be covered more than once
all_cells_covered= . *. /"1 tilings e. cells
no_cells_multi_covered= . *. /@~:"1 tilings

NB. tile solutions
tilings #~ all_cells_covered *. no_cells_multi_covered
)

NB. frequency distribution of numeric items
freqdist=: ~.@] ,: #/.~

gridslots=: 3 : 0
```

```
NB.*gridslots v-- checks rigid rotations of grid.
NB.
NB. Checks that the rigid rotations of the grid when partitioned
NB. and sorted do not produce a different set of slots than used
NB. by the tiling verbs.
NB.
NB. monad: blit =. gridslots uuIgnore

NB. reversals, rotations, and transposes of 4x4 grid
riggrids=. <"2 i."1 ] 4 4 , _4 4 , 4 _4 ,: _4 _4
riggrids=. ~. riggrids , |:&.> riggrids
rigslots=. /:~ ~. ; (/:~)"1&.> {{ > /:~ ,2 <"1 y }} &.> riggrids

NB. horizontal and vertical tile verb slots should match rigid slots
grid=. i. 4 4
slots=. > /:~ ,2 <"1 grid,|:grid
'slots do not match' assert slots -: rigslots

NB. if we allow a domino to split diagonally and
NB. act like a chess bishop that can only move one
NB. cell then we get more potential tiling slots
diagslots=. a: -.~ , ~. }."1 ,/ (<"1/.)&> riggrids
diagslots=. /:~ > ~. /:~&.> (<"0 ] _2 + #&> diagslots) }.&.> diagslots

NB. all 2x1 tile slots in 4x4 grid
rigslots;<diagslots
)
```

*NB. session manager output*

smoutput=: 0 0 \$ 1!:2&2

tiledominoes0=: 3 : 0

*NB.\*tiledominoes0 v-- place 2x1 domino tiles on a 4x4 grid.*

*NB.*

*NB. This lame verb was created to "prove" that you cannot cover a*

*NB. a 4x4 grid with corners 0,15 excluded by 2x1 domino shaped*

*NB. tiles. This is a problem in `[\_Cummings:2021aa\_]` page 7.*

*NB. The random slot picking often fails even when it is possible*

*NB. to cover the grid. Repeated executions usually finds a*

*NB. solution when possible.*

*NB.*

*NB. monad: tiledominoes0 ilXcells*

*NB.*

*NB. tiledominoes0 i.0 NB. use full grid*

*NB. tiledominoes0 0 15 NB. exclude corner cells*

*NB.*

*NB. NB. random slot filling succeeds about 30% on full grid*

*NB. tilefreq (#@>@(1%{)@tiledominoes0)%> 1000#<i.0*

*NB.*

*NB. NB. no successes for grid without corner cells*

*NB. tilefreq (#@>@(1%{)@tiledominoes0)%> 1000#<0 15*

*NB.*

*NB. NB. as more 2x1 slots are removed random filing works better*

*NB. tilefreq (#@>@(1%{)@tiledominoes0)%> 1000#<0 1 14 15 2 3 8 9*



```
grid=. i. 4 4                      NB. each cell numbered
slots=. /:~ ,2 <\ "1 grid, |:grid  NB. all 2x1 horizontal & vertical slots

NB. cover grid cells
cover=. {{ y #~ -. +./@ (x&e.) &> y }}

NB. a complete cover of a 4x4 grid - returns uncovered cell count - 0 here
NB. # 14 15 cover 6 10 cover 8 12 cover 4 5 cover 9 13 cover 7 11 cover 2 3 cover 0 1 cover slots

NB. random list item
rpick=. ] { ~ [: ? #

NB. exclude grid cells
if. #y do.
  'invalid grid cell(s)' assert y e. ,grid
  slots=. y cover slots
  grid=. y -.~ ,grid
end.

NB. tile count and covered cells
tiles=. 0 [ ccells=. 0$a:

NB. randomly cover available slots until no slots remain
while. #slots do.
  domino=. rpick slots
  NB. smoutput tiles [ smoutput slots [ smoutput domino
  ccells=. ccells,domino
```

```
slots=. (>domino) cover slots
tiles=. >:tiles
end.
```

```
NB. tiles used ; uncovered cells remaining
tiles;,(,grid) -. ;ccells
)
```

```
tiledominoes1=: 3 : 0
```

```
NB.*tiledominoes1 v-- 4x4 grid missing first/last cell cannot be
NB. tiled by 2x1 vh dominoes.
```

```
NB.
```

```
NB. Show there are no tilings of a 4x4 grid missing the first and
NB. last cells by vertical and horizontal 2x1 dominoes by testing
NB. all possible solutions. Brute force lacks elegance but, when
NB. feasible, gets the job done. Solves `[_Cummings:2021aa_]`
NB. page 7.
```

```
NB.
```

```
NB. monad: iaSolutions =. tiledominoes1 uuIgnore
```

```
NB. all 2x1 horizontal & vertical slots
```

```
grid=. i. 4 4
slots=. > /:~ ,2 <\ "1 grid,|:grid
```

```
NB. exclude corner slots
```

```
slots=. slots #~ -. +./ "1 slots e. 0 15
```

```

NB. cells to be covered
cells=. ~. ,slots

NB. a solution must cover 14 cells this takes 7 2x1 tiles
NB. generate all possible 7 tile slot combinations
tilings=. , "2 ((-:#cells) comb #slots) { slots

NB. a solution must cover all cells and
NB. no cell must be covered more than once
all_cells_covered=.      *./"1 tilings e. cells
no_cells_multi_covered=. *./@~:"1 tilings

NB. count number of solutions
+/ all_cells_covered *. no_cells_multi_covered
)

tiledominoes2=: 3 : 0

NB.*tiledominoes2 v-- count domino tilings of 4x4 grid missing (y) cells.
NB.
NB. monad: iaSolutions =. tiledominoes2 ilExcludeCells
NB.
NB.   tiledominoes2 0 15 NB. corners missing
NB.   tiledominoes2 i. 0 NB. no missing cells

NB. all 2x1 horizontal & vertical slots
grid=. i. 4 4
slots=. > /:~ ,2 <\ "1 grid, |:grid

```

```
NB. only even numbers of cells may be excluded
NB. as any tiling consists of an even number of cells
'cell number(s) invalid' assert (y e. ,slots) *. 0=2|#y

NB. exclude (y) cells
slots=. slots #~ -. +./"1 slots e. y

NB. cells to be covered
cells=. ~. ,slots

NB. generate all possible tile slot combinations
tilings=. , "2 ((-:#cells) comb #slots) { slots

NB. a solution must cover all cells and
NB. no cell must be covered more than once
all_cells_covered=. *./"1 tilings e. cells
no_cells_multi_covered=. *./@~:"1 tilings

NB. count number of solutions
+/ all_cells_covered *. no_cells_multi_covered
)

tiledominoes3=: 3 : 0

NB.*tiledominoes3 v-- list 4x4 tile solutions.
NB.
NB. monad: btilTiles =. tiledominoes3 ilExcludeCells
```

```
NB.
NB.   tiledominoes3 0 15  NB. corners missing
NB.   tiledominoes3 i. 0  NB. no missing cells

NB. all 2x1 horizontal & vertical slots
grid=. i. 4 4
slots=. > /:~ ,2 <\ "1 grid, |:grid
_2 <\ "1 slots findtilings y
)

tiledominoes4=: 3 : 0

NB.*tiledominoes4 v-- list 4x4 tile solutions allowing diagonal
NB. tiles.
NB.
NB. This verb is pushing up against the limits of what dumb let's
NB. test all the solutions can easily achieve. By adding diagonal
NB. tiles the solution possibilities explode. The worst case is
NB. finding all the tilings when no cells are removed. (8 comb
NB. 42) is 118,030,185. Surprisingly this completely unoptimized
NB. hack still works on my 32 gig pc and finds 280 distinct
NB. tilings.
NB.
NB. monad: btilTiles =. tiledominoes4 iExcludeCells
NB.
NB.   tiledominoes4 0 15  NB. no corners diagonal tiles have solutions
NB.   tiledominoes3 0 15  NB. original problem no solutions
NB.
```

*NB. tiledominoes4 i. 0 NB. no missing cells - worst case*

*NB. all 2x1 horizontal & vertical & diagonal tilings*

```
_2 <\ "1 (;gridslots 0) findtilings y
)
```

*NB. frequency sorted by uncovered cells*

```
tilefreq=: [: (] { "1~ /:@(0&({ ))) freqdist
```

*NB.POST\_TileDominoes post processor.*

```
(".;(0=nc <'SHOWSMO_ijod_'){ '1';'SHOWSMO_ijod_' ) smoutput IFACE=: (0 : 0)
```

```
NB. (TileDominoes) interface word(s): 20240521j161543
```

```
NB. -----
```

```
NB. tiledominoes0 NB. place 2x1 domino tiles on a 4x4 grid
```

```
NB. tiledominoes1 NB. 4x4 grid missing first/last cell cannot be tiled by 2x1 vh dominoes
```

```
NB. tiledominoes2 NB. count domino tilings of 4x4 grid missing (y) cells
```

```
NB. tiledominoes3 NB. list 4x4 tile solutions
```

```
NB. tiledominoes4 NB. list 4x4 tile solutions allowing diagonal tiles
```

```
)
```

```
cocurrent 'base'
```

```
coinsert 'TileDominoes'
```

# Index

assert, 5

comb, 5

findtilings, 5

freqdist, 6

gridslots, 6

IFACE, 14

IFACEWORDSTileDominoes, 4

ROOTWORDSTileDominoes, 4

smoutput, 8

tiledominoes0, 8

tiledominoes1, 10

tiledominoes2, 11

tiledominoes3, 12

tiledominoes4, 13

tilefreq, 14

VMDTileDominoes, 5