

Homework #1 - Tic Tac Toe

Prof. Marco Rivera

Students:

Christopher Arredondo - 2016094916

Erick Andrés Obregon Fonseca - 2016123157

Allan Navarro Brenes - 200943530



Department of Computer Engineering
Languages
May 2018

| | |
|---|-----------|
| User Manual | 2 |
| Start | 2 |
| Description of implemented functions | 5 |
| Interface | 5 |
| Matrix | 8 |
| Description of used data structures. | 15 |
| List | 15 |
| Matrix | 15 |
| Vector | 15 |
| Detailed description of developed algorithms | 16 |
| Known Problems | 18 |
| Activities by student. | 19 |
| Problems found | 20 |
| Conclusions and recommendations | 21 |
| References | 22 |

User Manual

Start

There are two ways to start the game:

The first way is with the GUI itself, giving it the rows and columns as parameters in “m” and “n”.

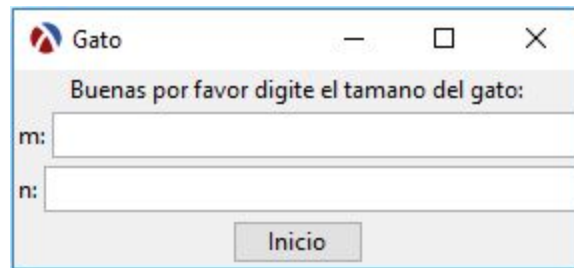



Figure 1. Getting the matrix dimensions.

The second way is with a CLI in which you give rows and columns as parameters.

```
> , (TTT 5 3)
```



| | | |
|----|----|----|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |
| 9 | 10 | 11 |
| 12 | 13 | 14 |

Figure 2. An example of 5 rows with 3 columns

When the user presses a button this will be changed to an "O" and the machine will put an "X" which is calculated using a greedy algorithm.



| | | |
|----|----|----|
| 0 | X | 2 |
| 3 | 4 | 5 |
| 6 | O | 8 |
| 9 | 10 | 11 |
| 12 | 13 | 14 |

Figure 3. An example of 5 rows with 3 columns An "X" and an "O".

The user or the machine wins if any of the two completes a diagonal, row or column.

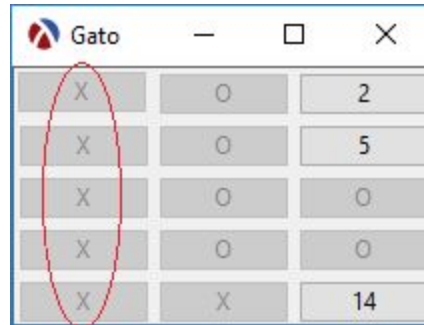


Figure 4. In this case, the user lost.

Finally when the user wins, loses or ties with the machine, these windows are returned respectively.

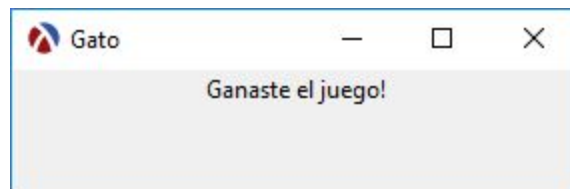


Figure 5. User wins.

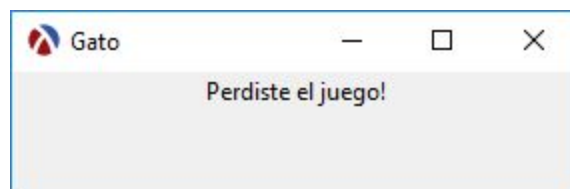


Figure 6. User loses.

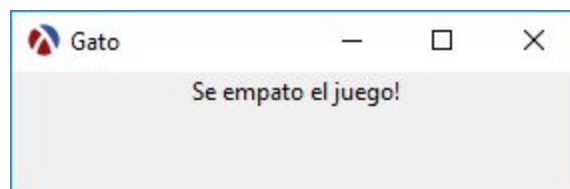


Figure 7. The user tied.

Description of implemented functions

To explain the implemented functions in a detailed and more efficient way, we will divide it in three parts, Interface, matrix functions and finally the greedy algorithm in the implemented algorithm section

Interface

The interface functions start with the CLI game initiator “TTT”, what it does is, in general terms, starts the game in a manual way.

```
(define (TTT mT nT)
  (set! m mT)
  (set! n nT)
  (set! listaPrincipal (poblarMatriz m n listaPrincipal 0))
  (inicio m n)
  (send frame show #t))
```

Figure 8. TTT function

Second set of functions are the frame functions, these functions are used to generate the required GUI elements to display buttons and strings.

```
;frame principal
(define frame
  (instantiate frame%
    ("Gato")))

;boton para iniciar juego
(new button% [parent intro]
  [label "Inicio"]
  [callback (lambda (button event)
    (send intro show #f)
    (set! m (string->number (send mText get-value)))
    (set! n (string->number (send nText get-value)))
    (set! listaPrincipal (poblarMatriz m n listaPrincipal 0))
    (inicio m n)
    (send frame show #t)))]))

(define intro (new frame%
  [label "Gato"]
  [width 300]
  [height 100]))

;texto
(define msg (new message% [parent intro]
  [label "Buenas por favor digite el tamaño del gato:"]))

;texto
(define mText (new text-field%
  [label "m:"]
  [parent intro]))

;texto
(define nText (new text-field%
  [label "n:"]
  [parent intro]))
```

Figure 9. Interface functions

Third comes functions to connect the matrix to the interface, the interface buttons being stored in a list have to be interpreted as a matrix.

```
(define (poblarMatriz m n matriz contador)
  (cond ((equal? (* m n) (length matriz))
    (set! matrizPrincipal matriz)
    matrizPrincipal)
    (else (poblarMatriz m n (append matriz (list contador)) (+ contador 1)))))
```

Figure 10. this function generates the matrix from the interface, based on the size given.

```

;coordenada a numero
(define (numToCoor pos i columnas)
  (cond ((< pos columnas)
        (list i pos))
        (else
         (numToCoor (- pos columnas) (+ i 1) columnas))))

(define (coorToNum coordenada columnas)
  (+ (* (car coordenada) columnas) (cadr coordenada)))

;funcion recursiva para pasar de una lista a una matriz
(define (listaMatriz mTemp nTemp lista temp resultado)
  (cond ((equal? nTemp 0)
        resultado)
        ((equal? mTemp 0)
         (listaMatriz n (- nTemp 1) lista '() (append resultado (list temp))))
        (else
         (listaMatriz (- mTemp 1) nTemp (cdr lista) (append temp (list (car lista))) resultado))))

```

Figure 11. these functions are to pass from coordinates to rows columns.

Forth, this function is one of the main ones, as it is where it connects to the greedy algorithm and generates all the game in general.

```

;creacion de botones y los eventos de cada uno
(define (inicio m n)
  (let ((panelGato (instantiate table-panel%
    (panelPrincipal)
    (style '(border))
    (dimensions (list m n))
    (column-stretchability (if (memq 1 '(0 1)) #t #f))
    (row-stretchability (if (memq 1 '(0 2)) #t #f)))))
    ;-----
    (let ((temp (for/vector ([j listaPrincipal])
      (new button%
        [parent panelGato]
        [label (format "~a" j)]
        [callback (lambda (button event)
          (send button enable #f)
          ;----aqui se agrega a la interfaz en la parte del usuario
          ;aqui se cambia por un cero en la interfaz
          (send button set-label "O")
          ;aqui se agrega a la matriz
          (set! posicionUsuario (numToCoor j 0 n))
          (set! matrizPrincipal (put matrizPrincipal 'O (car posicionUsuario) (cadr posicionUsuario)))
          (cond ((win? 'O matrizPrincipal)
                (send gana show #t)))
          ;aqui inicia el algoritmo goloso
          (cond ((null? (viable_candidates matrizPrincipal '(X O)))
                (send empate show #t))
                (else
                 (set! posicionMaquina (goloso matrizPrincipal '(X O)))

                 (set! temp (append temp (list posicionMaquina)))
                 (set! matrizPrincipal (put matrizPrincipal 'X (car posicionMaquina) (cadr posicionMaquina)))
                 (set! posicionButton (vector-ref buttons (coorToNum posicionMaquina n)))
                 (send posicionButton set-label "X")
                 (send posicionButton enable #f)
                 (cond ((win? 'X matrizPrincipal)
                       (send perdio show #t))
                       (cond ((null? (viable_candidates matrizPrincipal '(X O)))
                             (send empate show #t)))
                ))
                ))))))
    (set! buttons (vector-append buttons temp))
    (set! matrizPrincipal (listaMatriz n m listaPrincipal '() '()))))

```

Figure 12. Playing.

Finally there's the win, lost, tie windows that are created after the games ends.

```

;frame gano el juego
(define gano (new frame%
  [label "Gato"]
  [width 300]
  [height 100]))
(define msgGano (new message% [parent gano]
  [label "Ganaste el juego!"]))
;frame perdio el juego
(define perdio (new frame%
  [label "Gato"]
  [width 300]
  [height 100]))
(define msgPerdio (new message% [parent perdio]
  [label "Perdiste el juego!"]))
;frame nadie gano el juego
(define empate (new frame%
  [label "Gato"]
  [width 300]
  [height 100]))
(define msgEmpate (new message% [parent empate]
  [label "Se empato el juego!"]))

```

Figure 13. three types of game endings.

Matrix

The create-matrix function is used to create the matrix, which receives as an argument the number of rows and the number of columns and returns an array with the indicated dimensions. The algorithm is responsible for adding the m rows by calling an auxiliary function which creates a row of n elements and returning it to later add it to the matrix.


```

Create row
@param columns - row size
@return columns - size row
|#
(define (create-row columns)
  (cond
    ((zero? columns) '() )
    (else (append '(0) (create-row (- columns 1)))))
  )
)

#|
Create matrix
@param rows - rows number
@param columns - columns number
@return matrix of rows x columns
|#
(define (create-matrix rows columns)
  (cond
    ((zero? rows) '() )
    (else (append (list (create-row columns)) (create-matrix (- rows 1) columns))))
  )
)

```

Figure 14. Functions used to create a mxn matrix.

In order to make the modification of the matrix, two algorithms were created. The first of these is put, which receives the matrix to be modified, the value by which it is desired to change the original and the coordinates i and j where it is desired to replace the element. The algorithm runs through the rows until it reaches element i, once the row is found, it is called a change-row function which is responsible for putting the element in the indicated column j.

```

#|
Change row
@param value - value to change
@param row - row to change
@param j - position in the row
@return row with the value changed
|#
(define (change-row value row j)
  (cond
    ((null? row) '())
    ((zero? j) (append (list value) (change-row value (cdr row) (- j 1))))
    (else (append (list (car row)) (change-row value (cdr row) (- j 1))))
  )
)

#|
Put value in (i, j) position in matrix
@param matrix - matrix to change
@param value - value to change
@param i - position in rows
@param j - position in columns
@return matrix with value changed
|#
(define (put matrix value i j)
  (cond
    ((null? matrix) '())
    ((zero? i) (append (list (change-row value (car matrix) j)) (put (cdr matrix) value (- i 1) j)))
    (else (append (list (car matrix)) (put (cdr matrix) value (- i 1) j)))
  )
)

```

Figura 15. Functions used to change a matrix value.

The second one is get-value, which takes as an argument the matrix and the coordinates of the element that we want to obtain. The algorithm takes row i and column j and returns the value found in that position.

```

#|
Get value of the row
@param value - value to get
@param row - row to get
@param j - position in row
@return value
|#
(define (get-value-row row j)
  (cond
    ((zero? j) (car row))
    (else (get-value-row (cdr row) (- j 1)))
  )
)

#|
Get value of the (i, j) position in matrix
@param matrix - matrix to get value
@param i - position in rows
@param j - position in columns
@return value
|#
(define (get-value matrix i j)
  (cond
    ((zero? i) (get-value-row (car matrix) j))
    (else (get-value (cdr matrix) (- i 1) j))
  )
)

```

Figure 16. Functions used to get a matrix value.

To verify if there is a row that has already been filled, the check-rows function is used, which receives by parameters the matrix, the initial position of the rows and the element to be verified ("X" or "O") . The algorithm checks row by row if all the elements of that list have the same symbol by the function winner-row? which returns a Boolean value with respect to whether the indicated row has all the elements equal or not. The check-rows function returns the number of the row where the winner of the game was found.

```

#|
Check if there is a winner row
@param row - row to check
@param symbol - symbol to check in all positions
@return boolean
|#
(define (winner-row? row symbol)
  (cond
    ((null? row) #t)
    ((equal? symbol (car row)) (winner-row? (cdr row) symbol))
    (else #f)
  )
)

#|
Check if there is a winner row
@param matrix - matrix to check
@param pos - actual position
@param symbol - symbol to check in all positions
@return winner row position or () if it doesn't exist
|#
(define (check-rows matrix pos symbol)
  (cond
    ((null? matrix) '())
    ((winner-row? (car matrix) symbol) pos)
    (else (check-rows (cdr matrix) (+ pos 1) symbol))
  )
)

```

Figure 17. Functions used to check a winner row.

When you want to check if there is a winning column, check-columns are used. This function receives as arguments the matrix where you want to verify, the initial position of the columns and the symbol that you want to verify if it was a winner (X or O). The algorithm is responsible for checking column by column if the element in the position indicated in that list has the same symbol by the function winner-column? which returns a Boolean value with respect to whether the element in the indicated column has the same symbol as the indicated one. The check-columns function returns the position of the column where the winner of the game was found.

```

#|
Check if there is a winner column
@param column - column to check
@param symbol - symbol to check in all positions
@param row - position in row
@param column - actual position in column
@return boolean
|#
(define (winner-column? matrix symbol row column)
  (cond
    ((equal? row (length matrix)) #t)
    ((equal? symbol (get-value matrix row column)) (winner-column? matrix symbol (+ row 1) column))
    (else #f)
  )
)

#|
Check if there is a winner column
@param matrix - matrix to check
@param pos - actual position
@param symbol - symbol to check in all positions
@return winner column position or () if it doesn't exist
|#
(define (check-columns matrix pos symbol)
  (cond
    ((equal? pos (length (car matrix))) '())
    ((winner-column? matrix symbol 0 pos) pos)
    (else (check-columns matrix (+ pos 1) symbol))
  )
)

```

Figure 18. Functions used to check a winner column.

If you want to check if there is a winning diagonal, `check-diagonals` function is used. This function receives as arguments the matrix where you want to verify, the initial position of the diagonals and the symbol that you want to verify if it was the winner (X or O). The algorithm is responsible for checking diagonal by diagonal (having as a condition of stop that what is going to be analyzed is a diagonal) if the element in the coordinate *i, j* has the same symbol by the function `winner-diagonal?` which takes as arguments the matrix, the coordinate *i, j* and the symbol and returns a Boolean value with respect to whether the element in the indicated coordinate has the same symbol as that indicated in the parameters. The `check-diagonals` function returns the position where the diagonal starts where the winner of the game was found.

```

#|
Winner diagonal
#|
(define (winner-diagonal matrix row column symbol)
  (cond
    ((or (equal? (length matrix) row) (equal? (length (car matrix)) column)) #f)
    ((equal? (get-value matrix row column) symbol)
     (winner-diagonal matrix (+ row 1) (+ column 1) symbol))
    (else #f)
  )
)

#|
Check if there is a winner diagonal
@param matrix - matrix to check
@param diagonal - actual position
@param symbol - symbol to check in all positions
@return winner diagonal position or () if it doesn't exist
#|
(define (check-diagonals matrix diagonal symbol)
  (cond
    ((zero? (+ diagonal 1)) '())
    ((winner-diagonal matrix diagonal 0 symbol) diagonal)
    (else (check-diagonals matrix (- diagonal 1) symbol))
  )
)

```

Figure 19. Functions used to check a winner diagonal

To check if there is any winning reverse diagonal, the `check-diagonals-inverse` function is used. This function receives as arguments the matrix where you want to verify, the initial position of the diagonals and the symbol that you want to verify if it was the winner (X or O). The algorithm is responsible for checking diagonal by diagonal (having as a condition of stop that what is going to be analyzed is a diagonal) if the element in the coordinate *i, j* has the same symbol by the function `winner-diagonal?` which takes as arguments the matrix, the coordinate *i, j* and the symbol and returns a Boolean value with respect to whether the element in the indicated coordinate has the same symbol as that indicated in the parameters. The `check-diagonals` function returns the position where the diagonal starts where the winner of the game was found.

```

#|
Winner diagonal reverse
#|
(define (winner-diagonal-reverse matrix row column symbol)
  (cond
    ((or (equal? row -1) (equal? (length (car matrix)) column)) #t)
    ((equal? (get-value matrix row column) symbol)
     (winner-diagonal-reverse matrix (- row 1) (+ column 1) symbol))
    (else #f))
  )
)

#|
Check if there is a winner reverse diagonal
@param matrix - matrix to check
@param diagonal - actual position
@param symbol - symbol to check in all positions
@return winner reverse diagonal position or () if it doesn't exist
#|
(define (check-diagonals-reverse matrix diagonal symbol)
  (cond
    ((equal? diagonal (- (length (car matrix)) 2)) '())
    ((winner-diagonal-reverse matrix diagonal 0 symbol) diagonal)
    (else (check-diagonals-reverse matrix (- diagonal 1) symbol))
  )
)

```

Figure 20. Functions used to check a winner reverse diagonal.

Description of used data structures.

List

Lists were used in various parts of the project, including the interface and the matrix.

Matrix

The use of the matrix is itself the core of the project and we implemented various matrix functions.

Vector

Vectores were used in the process of creating the interface, mainly for the buttons.

Detailed description of developed algorithms

This section describes the main algorithms implemented for the project.

For the case of the greedy algorithm, the set of functions programmed for the matrices explained above is used and the objectives of the greedy algorithm are added to it.

```
#lang racket
(require "funciones.rkt")
(require racket/include)
(provide (all-defined-out))

(define (goloso matrix symbols)
  (solution (objetivo (viable_candidates matrix symbols) matrix symbols) )
)
```

Figure 22. Inclusion of the functions and definition of the main method.

The function "greedy" is called with the current matrix of the game and a list of two elements with the symbols that are going to be used by the computer and the user respectively, example: '(X O). This function executes the different objectives of the greedy algorithm in order to finally return the optimal solution at that moment.

First it is necessary to choose from among the matrix those candidates that are viable to be used in the solution, that is, those positions that are not already used. This is done by the function viable_candidates:

```
10 ;Funcion de Viabilidad
11 ;función que determina los indices de la matrix de los candidatos a solución que son viables
12 ; matrix: representa el gato
13 ; symbols los symbols usados para marcar la posición escogida por los usuarios ej. '(X O)
14 (define (viable_candidates matrix symbols)
15   (viable_candidates_aux matrix symbols '() 0 0)
16 )
17 ; función auxiliar para determinar los candidatos viables
18 ; n fila
19 ; m columna
20 (define (viable_candidates_aux matrix symbols positions m n)
21   (cond((null? matrix) positions)
22         ((null? (car matrix))
23          (viable_candidates_aux (cdr matrix) symbols positions 0 (+ n 1)))
24         ((and (different? (caar matrix) (car symbols)) (different? (caar matrix) (cadr symbols)))
25          (viable_candidates_aux (append (list (cadr matrix)) (cdr matrix)) symbols (append positions (list (list n m)) (+ m 1) n))
26          (viable_candidates_aux (append (list (cadr matrix)) (cdr matrix)) symbols positions (+ m 1) n))
27         (else
28          (viable_candidates_aux (append (list (cadr matrix)) (cdr matrix)) symbols positions (+ m 1) n))
29         )
30   )
31 )
```

Figure 23.

In this function, the matrix is traversed and each position is evaluated to see if it is available, those that are actually available are stored in a list of pairs (i j) that represent the position in the matrix.

After this, the objective function is performed, which is responsible for putting a note to each position of the list of viable candidates. The highest note is assigned to the positions in which a computer win would occur in the next movement, the next priority is the positions in which the user would win if he placed the mark in the next movement. Finally, priority is given to those positions in which in a column or a row there are markers of the opponent and none of the computer to avoid the formation of columns or rows. This function produces a list with pairs in the following way: ((i j) note).

```

49 ;Funcion Objetivo
50 ; devuelve una lista donde cada par se evalua, cada elemento tiene la siguiente estructura ai= ((fila columna) nota)
51 (define (objetivo candidates matrix symbols)
52   (objetivo_aux candidates '() matrix symbols)
53 )
54
55 (define (objetivo_aux candidates rated_candidates matrix symbols)
56   (cond ( (null? candidates)
57         rated_candidates)
58         (else
59          (objetivo_aux (cdr candidates) (append rated_candidates (list (list (car candidates) (rate (car candidates) matrix symbols)))) matrix symbols)
60          )
61         )
62 )
63
64 ;función que evalúa un candidato dentro de la matrix del juego
65 ;@param coordinates : coordinates (i j) a rate
66 (define (rate coordinates matrix symbols)
67   (cond ( (hypothetical_win? coordinates matrix (car symbols)) 1000);computer wins
68         ( (hypothetical_win? coordinates matrix (cadr symbols)) 500);player wins
69         ( (or (and (exists_in_col? coordinates matrix (cadr symbols)) (not (exists_in_col? coordinates matrix (car symbols))))
70              (and (exists_in_row? coordinates matrix (cadr symbols)) (not (exists_in_row? coordinates matrix (car symbols))))
71              (exists_in_diagonal? coordinates matrix (cadr symbols)))
72           ) 100)
73         (else 50)
74   )
75 )
76 )
77 )

```

Figure 24. Objective function.

The solution function takes the list produced by the objective and gets the position that has the highest priority at that moment (the one with the highest grade), this is the position that is returned to the interface so that it is placed in the matrix of the game.

```

34 (define (solution evaluated_candidates)
35   (solution_aux (car evaluated_candidates) (cdr evaluated_candidates))
36 )
37
38 (define (solution_aux best_candidate candidates)
39   (cond ( (null? candidates)
40         (car best_candidate) )
41         ((< (cadr best_candidate) (cadr candidates))
42          (solution_aux (car candidates) (cdr candidates)) )
43         (else
44          (solution_aux best_candidate (cdr candidates)) )
45   )
46 )

```

Figure 25. Solution function

Known Problems

- After conducting tests in the game, it was found that on isolated occasions, the computer loses the game. The problem is identified in the way that the greedy algorithm assesses the candidates for solution and at the time of the execution of this document, work is being done to resolve this situation.

Activities by student.

A table is shown below to resume the main activities and which member of the team was in charge of them.

| Task | Description | Responsible | Duration (hours) | Date |
|------|---|-------------|------------------|-----------|
| 0 | First meeting, assigning tasks to the members of the team | All | 1 | 27/4/2018 |
| 1 | Functions to work with matrixes and to check columns, rows o diagonals in the TTT | Erick | 5 | 30/4/2018 |
| 2 | GUI | Christopher | 5 | 3/5/2018 |
| 3 | Greedy Algorythm | Allan | 3 | 7/5/2018 |
| 4 | Project integration, debugging and tests | Allan | 2 | 9/5/2018 |
| | | | 16 | |

Problems found

- When the function was called to validate if there was a winning column, it failed when the last column was the winner. It was found that the problem was that the n columns were not being taken, but rather, the m rows of the matrix. To solve it, we simply change the m rows indicated by the n columns and with that the algorithm works correctly.
- Another known problem was that in the reverse diagonal diagonals were being taken that did not meet the conditions to be taken into account. This was because the stop condition was incorrect and was corrected by placing the function to stop when it found a diagonal that did not have the same amount of elements as columns.
- It was also detected that when the matrix was created it was created in an inverse manner, that is, instead of creating an $m \times n$ matrix, an $n \times m$ matrix was created, so all the positions were taken wrongly. To solve this, the order in which the correct parameters were received was simply changed.
- When a button was pressed instead of changing the element in the coordinate (i, j) it was done in the coordinate (j, i) , so to solve the problem, only the order of the coordinate was inverted.

Conclusions and recommendations

- It was possible to use purely recursive algorithms to implement a greedy algorithm and find different solutions for the Tic Tac Toe game.
- It was possible to divide the project in a modular way, so that the members could work individually in each part and in conclusion "exposing" an interface to those who require their functions. The division of tasks is a fundamental problem in any project and the success of this project is due to the previous planning work that was carried out.
- It was possible to integrate the interface with the algorithm to show the functionality of the game

References

- 4.9 Pairs and Lists. (2018). Docs.racket-lang.org. Retrieved 1 May 2018 from <https://docs.racket-lang.org/reference/pairs.html>
- 4.11 Vectors. (2018). Docs.racket-lang.org. Retrieved 3 May 2018, from <https://docs.racket-lang.org/reference/vectors.html>
- Table Panel. (2018). Docs.racket-lang.org. Retrieved 3 May 2018, from <https://docs.racket-lang.org/table-panel/index.html>