Tarea 1: Documentación

Paradigma Funcional: 4Line

José Julián Camacho Hernández

Jose Fabián Mendoza Mata

Sergio Andrés Ríos Campos

Área de Ingeniería en Computadores, Instituto Tecnológico de Costa Rica

CE-3104: Lenguajes, Compiladores e Intérpretes

Profesor: Marco Rivera Meneses

30 de setiembre de 2020

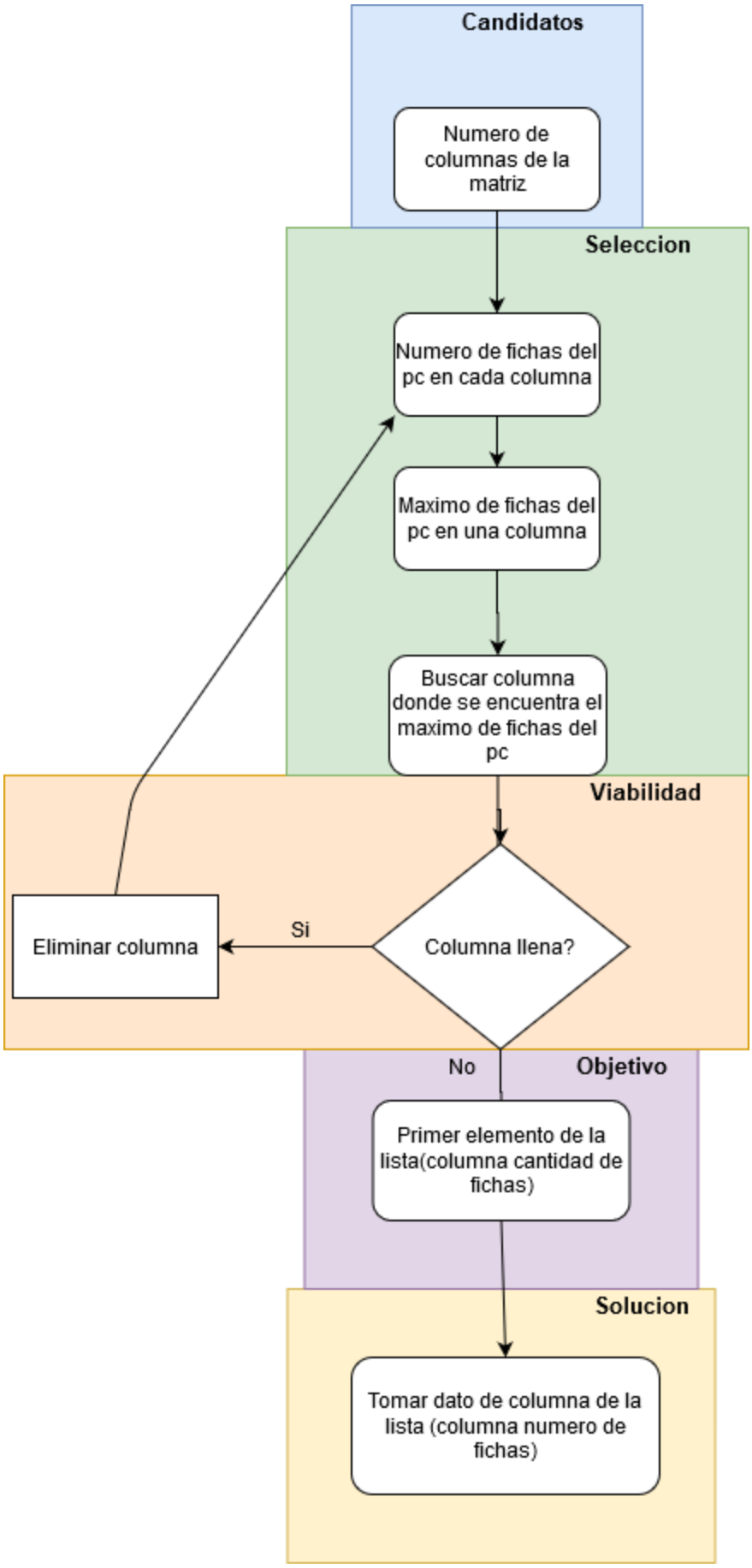
Tabla de contenidos

1. Breve descripción del proyecto

El 4 en Línea pertenece a la familia de juegos populares que ofrece gran entretenimiento. Para efectos de este proyecto, se jugará contra la computadora. El objetivo de este juego consiste en colocar cuatro fichas en una fila continua vertical, horizontal o diagonalmente.

1.1. Descripción detallada del algoritmo de solución desarrollado.

- **Greedy Algorithm(algoritmo codicioso)**
    1. **Función candidatos**: El objetivo de esta función es obtener los candidatos que se introducirán al algoritmo para llegar a una solución. En este caso, los candidatos serán todas las columnas de la matriz a jugar (Ancho de la matriz del juego)
    2. **Función Selección:** Esta función se encarga de evaluar y seleccionar las posibles soluciones que más se adecúen a la resolución del problema. Se utilizarán las columnas donde haya mayor cantidad de fichas de la pc, ya que aumenta la posibilidad de formar 4 en línea. Esta función necesita de otras funciones como “compu?” que se utiliza para saber si una ficha es del jugador o del pc, “maximo” cuyo objetivo es obtener el máximo número de fichas en una columna y “buscar” la cual busca la columna en donde el número retornado por máximo, coincide con el número de fichas en esa columna. Al final de la función de selección, se retornará una matriz con listas donde sus elementos serán el número de columna y la cantidad de fichas del pc que contienen, ejemplo ((0 2) (1 0) (3 1))
    3. **Función Viabilidad:** Esta función se encarga de verificar si las soluciones seleccionadas pueden aplicarse en el momento de ejecutar el algoritmo. En este trabajo se verificará si la columna seleccionada se encuentra llena, es decir, no se podrían colocar más fichas. En caso de ser verdadero, la columna es eliminada del algoritmo y se repite el proceso de selección .
    4. **Función Objetivo:** Es la encargada de evaluar las soluciones presentadas y asegurarse de elegir la que más se adecúe a las necesidades del momento. Para este proyecto se verificarán aspectos como cantidad de soluciones disponibles y cantidad de fichas del oponente en la columna seleccionada. Al finalizar este proceso se retornará una lista con la columna y la cantidad de fichas en esa columna.
    5. **Función Solución:** Es la encargada de dar el paso final y retornar la solución a utilizar. En este caso, se retornará el número de columna(0-16) a depositar la ficha. Al terminar el algoritmo, será turno el del jugador de colocar la ficha, posteriormente, se repetirá el proceso.


Candidatos
Numero de columnas de la matriz
Seleccion
Numero de fichas del pc en cada columna
Maximo de fichas del pc en una columna
Buscar columna donde se encuentra el maximo de fichas del pc
Viabilidad
Columna llena?
Si
Eliminar columna
No
Objetivo
Primer elemento de la lista(columna cantidad de fichas)
Solucion
Tomar dato de columna de la lista (columna numero de fichas)

1.2. Descripción de las funciones implementadas.

Lógica:

`1- (hayCeros? lst)`

- Verifica si hay ceros en la lista, en este caso verifica si está vacía la fila.
- Entradas: lst: lista donde se buscan los ceros.
- Salidas: #t si hay al menos un cero, #f si no existe alguno.

`2- (full matrx)`

- Recorre todas las columas de la matriz y verifica si la matriz está llena, para determinar si se dio un empate.
- Entradas: matrx: matriz por verificar si está llena.
- Salidas: #t si no hay ceros en ninguna fila, #f si hay ceros en alguna fila.

`3- (validDim? rows col)`

- Verifica si las dimensiones están en el rango.
- Entradas: rows: número de filas, col: número de columnas.
- Salidas: #t si ambas entradas están entre 8 y 16, #f si no lo cumplen.

`4- (makeRow col_number)`

- Hace una lista de tantos ceros como número de columnas se especifiquen
- Entradas: col_number: número de columnas.
- Salidas: lista con ceros.

`5- (generateMatrx m n)`

- Generar una matriz de dimensiones m x n vacía (solo de ceros).
- Entradas: m: número de filas, n: número de columnas de la matriz.
- Salidas: matriz m x n llena de ceros.

`6- (getPos elem lst)`

- Obtener el índice de un elemento en una lista.
- Entradas: elem: elemento por buscar, lst: lista donde se busca.
- Salidas: posición del elemento en la lista.

`7- (at index lst)`

- Obtener el elemento en ese índice.
- Entradas: index: posición en la lista por obtener, lst: lista donde se busca.
- Salidas: elemento en ese índice.

`8- (len lst)`

- Cuenta los elementos de una lista.

- Entradas: lst: lista por verificar su largo.
- Salidas: longitud de una lista.

9- `(doFunc func lista)`

- Realiza una función en cada elemento de una lista.
- Entradas: func: función por realizar, lista: lista donde se va a aplicar la función.
- Salidas: lista con la función aplicada.

10- `(traspuesta matrx)`

- Devuelve la traspuesta de una matriz.
- Entradas: matrx: matriz para calcular su traspuesta.
- Salidas: matriz traspuesta de la matriz dada.

11- `(fullColumn? colNum matrx)`

- Recorre una columna y verifica si una columna está llena.
- Entradas: colNum: número de columna por verificar, matrx: matriz donde se verifica.
- Salidas: #f si hay ceros en esa columna, lo que indica que no está llena; #t si hay al menos un cero.

12- `(gravityCheck column player)`

- Verifica el efecto que una ficha cae. Si el siguiente elemento en la lista (siguiente ficha) es diferente de cero, implica que la ficha del jugador debe ir en el espacio actual. Si el siguiente elemento es nulo (el actual es el último espacio de la lista) la ficha debe colocarse ahí.
- Entradas: player: ficha de quien va a jugar:1 (jugador), 2 (máquina);
- Salidas: lista con la ficha del jugador en la ubicación esperada por la gravedad.

13- `(4Line? player line cont)`

- Recorre una línea y verifica si hay 4 elementos seguidos del mismo número en una línea.
- Entradas: player: ficha de quien va a jugar :1 (jugador), 2 (máquina); line: fila (lista) donde se va a revisar; cont: contador para llevar control de cuántos elementos del mismo tipo están seguidos.
- Salidas: #t si hay 4 elementos seguidos en una fila, #f si no existen 4.

14- `(checkAllLines player matrx)`

- Recorre la matriz y verifica en todas las líneas si hay 4 en línea.
- Entradas: player: ficha de quien va a jugar:1 (jugador), 2 (máquina); matrx: matriz actual por recorrer para verificar.

- Salidas: #t si en alguna línea de la matriz hay 4 en línea, #f si la matriz llega a ser nula, lo que implica que no existen 4 en línea.

15- `(recorrerDiag_Sup_L_R_aux matrx i j vec result)`
- Recorrer la matriz superior (espacios por encima de la diagonal principal de la matriz) por sus diagonales de izquierda(L) a derecha (R).
- Entradas: matrx: matriz actual por recorrer para verificar.
- Salidas: una lista de listas, donde cada una de las sublistas contienen los valores de los elementos en las diagonales de izquierda a derecha ascendentemente, en la parte superior a la diagonal principal de la matriz.

16- `(recorrerDiag_Inf_L_R_aux matrx i j vec result)`
- Recorrer la matriz inferior (espacios por debajo de la diagonal principal de la matriz) por sus diagonales de izquierda(L) a derecha (R).
- Entradas: matrx: matriz actual por recorrer para verificar.
- Salidas: una lista de listas, donde cada una de las sublistas contienen los valores de los elementos en las diagonales de izquierda a derecha ascendentemente, en la parte inferior a la diagonal principal de la matriz.

17- `(reverseList lista)`
- Invertir los elementos de una lista.
- Entradas: una lista
- Salidas: lista con sus elementos invertidos.

18- `(reverseRows matrx)`
- Invertir los elementos de las filas de la matriz (Invertir columnas).
- Entradas: matrx: matriz actual para invertir sus columnas.
- Salidas: una matriz con sus columnas invertidas, es decir, la columna cero pasa a ser la última columna y viceversa.

19- `(recorrerDiag_L_R matrx)`
- Recorrer la matriz por sus diagonales de izquierda a derecha. Unifica las funciones de diagonal superior e inferior.
- Entradas: matrx: matriz actual por recorrer para verificar.
- Salidas: una lista con sublistas que contienen todas las diagonales de izquierda a derecha de forma ascendente de una matriz.

20- (`recorrerDiag_R_L matrx`)
- Recorrer la matriz por sus diagonales de derecha a izquierda.
- Entradas: matrx: matriz actual por recorrer para verificar. Uneifica las funciones de diagonal superior e inferior con las columnas invertidas

- Salidas: una lista con sublistas que contienen todas las diagonales de izquierda a derecha de forma descendente de una matriz.

21- `(checkAllDiag player matrx)`

- Verifica si hay 4 en línea en las diagonales de izquierda a derecha o viceversa.
- Entradas: player: ficha de quien va a jugar:1 (jugador), 2 (máquina); matrx: matriz actual por recorrer para verificar.
- Salidas: #t si existen 4 en línea en alguna de las diagonales; #f si no las hay.

22- `(win? matrx)`

- Verifica quién ganó o si nadie ha ganado. Para ello, verifica si hay 4 en línea en filas, columnas y diagonales, para el jugador y para la máquina.
- Entradas: matrx: matriz actual por recorrer para verificar.
- Salidas: strings indicando quién ha ganado, o si no existe un ganador aún.

23- `(play player colNum matrx)`

- Función para realizar una jugada. Primero verifica si existe un ganador, si existe un empate o si la columna donde se va a jugar está llena. De no ser así, se coloca la ficha de acuerdo con la gravedad.
- Entradas: player: ficha de quien va a jugar :1 (jugador), 2 (máquina); colNum: número de columna donde se quiere tirar; matrx: matriz actual donde se va a realizar la jugada.
- Salidas: matriz con la jugada aplicada, o strings si no se puede realizar la jugada.

24- `(candi matrx)`

- Obtener el número de columnas de la matriz
- Entradas: Matriz del juego
- Salidas: número de columnas disponibles de la matriz

25- `(selec matrx index out exp)`

- Obtener la columna de la matriz con más fichas del pc
- Entradas: matrx: matriz del juego. index: índice de columnas que lo provee la interfaz. out: lista con las columnas y la cantidad de fichas del pc en cada columna. exp: lista de columnas llenas
- Salida: lista out

26- `(compu? lista index cant)`

- Verifica si fichas de cada columna de la matriz, son fichas del pc

- Entradas: lista: cada columna de la matriz. index: índice de columnas que lo provee la interfaz cant: cantidad de fichas del pc en cada columna

27- `(maximo mat)`

- Busca el máximo número de fichas que hay en cada columna de la matriz
- Entradas: matriz de tipo columna - número de fichas: ((0 0) (1 2) (2 0))
- Salidas: número máximo de fichas en las columnas

28- `(buscar mat elemento out)`

- Busca la columna donde esté el número que retorna la funcion maximo
- Entradas: mat: matriz de tipo columna - número de fichas:((0 0) (1 2) (2 0)) elemento: número retornado en función maximo out: salida con las columnas donde este el número de la función maximo
- Salidas: matriz tipo columna - número de fichas, con solo las columnas donde esté la mayor cantidad de fichas del pc

29- `(viabilidad columnas marx exp)`

- Verifica si las columnas seleccionadas en la función selec se encuentran llenas
- Entradas: columnas: matriz tipo columna - número de fichas retornadas en la función selec matrx: matriz del juego. exp: lista de columnas llenas
- Salidas: matriz tipo columna - número de fichas, sin las columnas llenas

30- `(eliminarCol elemento index lista)`

- Elimina una columna de la lógica del algoritmo si esta está llena
- Entradas: elemento: columna a eliminar. index: numero de columna a eliminar. lista:Matriz del juego
- Salidas: matriz sin la columna dada

31- `(objetivo soluciones matrx)`

- Busca la mejor solución posible entre las columnas dadas
- Entradas: soluciones: matriz tipo columna - número de fichas. matrx: matriz del juego
- Salidas: lista tipo columna - número de fichas con la solución más adecuada

32- `(solucion respuesta)`

- Saca la columna en la que el pc pondrá su ficha
- Entradas: lista tipo columna - número de fichas retornada en la función objetivo
- Salidas: número de columna a jugar

1.3. Descripción de la ejemplificación de las estructuras de datos desarrolladas.

Se desarrolló como estructura de datos principal una matriz.

((2 0 0 0 0 0 0 0)
(1 2 0 0 2 0 0 0)
(1 1 2 0 1 0 0 0)
(2 1 1 2 1 0 0 0)
(1 2 1 2 2 1 1 1)
(1 1 2 2 1 1 2 2)
(1 2 1 1 2 2 1 1)
(1 1 2 2 1 1 2 2))

El tablero del juego de 4Line fue modelado como una matriz. En Racket, una matriz se representa como una lista de listas. Para efectos de este proyecto, cada una de las sublistas equivale a cada una de las filas de la matriz, que reflejan cada una de las filas del tablero.

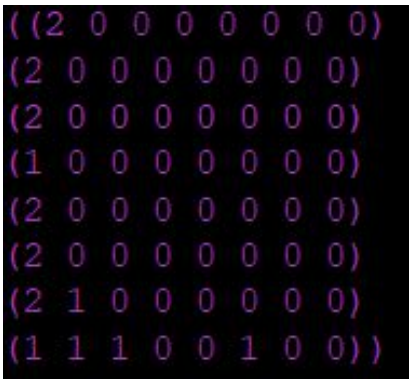
De esa manera, cada una de las columnas del tablero serán representadas en la matriz como los elementos de todas las filas que tengan la misma posición. Por ejemplo, la columna cero corresponderá a todos los primeros elementos de cada sublista. Por simplicidad, se utilizará la matriz traspuesta, para utilizar columnas como filas.

A cada elemento le corresponderá una ubicación correspondiente a la fila y columna donde se encuentra. Los elementos en la matriz, en este proyecto, serán de tres tipos:

- 0: significa que no se encuentra alguna ficha en esa posición, se representará en la interfaz como un espacio en blanco.

- 1: representa que en esa posición se encuentra una ficha del jugador, en la interfaz se mostrará como una ficha de color rojo.

- 2: corresponde a los espacios donde existe una ficha del computador, en la interfaz se representará como una ficha de color azul.



1.4. Problemas sin solución: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.

De manera general, se logró implementar la totalidad del proyecto y fue posible solucionar la mayoría de los problemas que fueron detectados.

Sin embargo, perduró un error al terminar el juego. Al finalizar, se detecta un ganador, pero no se cierra de inmediato la ventana de juego, por lo que pueden realizarse jugadas posteriores que pueden desencadenar errores.

1.5. Problemas encontrados: descripción detallada, intentos de solución sin éxito, soluciones encontradas con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.

i) **Verificación de 4 fichas en línea en diagonal:** la verificación de 4 en línea fue un proceso que conllevó varias horas de trabajo. Esto debido a que se necesitaban todas las diagonales de la matriz en listas, para ser evaluadas como si fueran una fila o una columna. Para solucionarlo fue necesario investigar métodos en Internet. Se consultó el punto uno de la bibliografía que plantea un algoritmo para obtener las diagonales de izquierda a derecha de manera ascendente de la matriz. Este fue totalmente adaptado al paradigma funcional. Sin embargo, la función no cubría el objetivo completamente, ya que se necesitaba verificar en las diagonales de derecha a izquierda de forma descendente. Debido a ello se tuvo que implementar una función para invertir las columnas de la matriz y de esa manera se resolvió el problema, reutilizando así el algoritmo anteriormente mencionado.

ii) **Verificación de ganador para matrices con mayor número de columnas que filas:** se presentó un problema al verificar 4 fichas en línea en las diagonales para matrices con $n > m$. Esto fue producido por el alcance del algoritmo explicado en el punto i). Para solucionar este problema, se implementó la idea de utilizar la matriz traspuesta para estos casos, ya que de esa manera, en esta las filas pasarían a ser columnas y viceversa. Debido a ello, para la nueva sería posible la verificación.

1.6. Plan de Actividades realizadas por estudiante: Planeamiento de las actividades, descripción de la tarea, tiempo estimado de completitud, responsable a cargo y fecha de entrega.

Tarea	Tiempo Estimado	Responsable	Fecha de entrega
Generación del tablero de juego	1 h	Julián Camacho	19/09/2020
Función candidatos del algoritmo codicioso	1h	Sergio Ríos	20/09/2020
Efecto de las fichas de caer	2.5 h	Julián Camacho	20/09/2020
Representación del tablero de juego en interfaz	5	José Fabián Mendoza Mata.	20/09/2020
Presentación preliminar de interfaz funcional	3	José Fabián Mendoza Mata	20/09/2020
Verificación de que se produjo un empate	4 h	Julián Camacho	21/09/2020
Interfaz funcional, el jugador juega y se ven sus fichas	2	José Fabián Mendoza Mata	21/09/2020
Función de selección del algoritmo codicioso	5h	Sergio Ríos	23/09/2020

Verificación de 4 fichas en línea recta de un mismo jugador	4 h	Julián Camacho	23/09/2020
Se pueden representar los movimientos de la máquina en la interfaz	3	José Fabián Mendoza Mata	23/09/2020
Función de viabilidad del algoritmo codicioso	3h	Sergio Ríos	24/09/2020
Función de objetivo del algoritmo codicioso	3h	Sergio Ríos	24/09/2020
Función Solución del algoritmo codicioso	1h	Sergio Ríos	24/09/2020
Verificación de 4 fichas en cualquier diagonal	6 h	Julián Camacho	25/09/2020
Habilitar la opción de jugar tanto para el usuario como la máquina	3 h	Julián Camacho	27/09/2020

1.7. Conclusiones.

i) Se logró crear una aplicación que modela el popular juego de 4 en línea utilizando Racket.

ii) Se aplicaron los conceptos y destrezas aprendidas durante el estudio del paradigma de programación funcional, tales como la sintaxis de Racket, la recursividad y la no utilización de variables ni ciclos iterativos.

iii) El problema fue modelado y resuelto utilizando matrices. Debido a ello, se implementaron diversas funciones donde se manipulaban las listas que componen la matriz. Entre ellas, funciones para recorrer la matriz por sus filas, por sus diagonales, la matriz traspuesta, conteo de elementos, entre otras.

iv) Se logró implementar un algoritmo tipo codicioso(greedy) que logra jugar contra el usuario de la aplicación, obteniendo columnas donde sea más adecuado colocar la ficha

1.8. Recomendaciones.

a. Se recomienda leer el manual de usuario antes de jugar, esto con el fin de comprender todos los procesos y partes que componen el programa.
b. Como es bien sabido, el programa utiliza un algoritmo codicioso para reproducir las jugadas del pc, al ser un algoritmo que, por diseño, consume grandes recursos. Por esto, se recomienda tener estos recursos a disposición al momento de ejecutar este programa.
c. La matriz del juego se puede expandir a la medida de 16x16, sin embargo, los recursos del pc y la extensión de la pantalla del mismo, hacen que la experiencia al jugar el 4 Line no sea la esperada, se recomienda usar medidas como 8x8, 9x9, 10x10.
d. Se recomienda evitar el uso de varias aplicaciones al mismo tiempo, ya que el algoritmo que utiliza la computadora para hacer jugada consume muchos recursos, al tener otros programas abiertos pueden afectar el desempeño del mismo.
e. Jugar en pantalla minimizada, ya que la librería nativa de racket se comporta de forma extraña en pantalla completa, a veces quitando widgets de la interfaz, también se recomienda evitar cambiar las dimensiones de la ventana.

1.9. Bibliografía consultada en todo el proyecto

Recorrido en diagonal de matriz - Curso de algoritmia. (2020). Recuperado 20 septiembre 2020, de https://sites.google.com/site/ticslearn/Recorrido_en_diagonal_de_matriz

H., Migeed, Z., & Migeed, Z. (2016). *How can I import a function from a file in racket?*. Recuperado el 20 septiembre 2020, de https://stackoverflow.com/questions/34756477/how-can-i-import-a-function-from-a-file-in-racket/34756648

2. Bitácora en digital, donde se describen las actividades realizadas.

Fecha	Estudiante(s)	Actividad
19/09/2020	Todos	Primera reunión de organización y división del trabajo correspondiente a cada uno de los integrantes de acuerdo con la cantidad de créditos en curso y la dificultad estimada de este.
19/09/2020	Julián Camacho	Se realizaron las primeras funciones de la lógica del juego. Se tomó la decisión de modelar el problema por medio de una matriz.
19/09/2020	Julián Camacho José Fabián Mendoza Mata	Se implementaron las funciones para generar la matriz de juego, y para lograr insertar una ficha en la misma y que tenga el efecto de caer.
19/09/20	Sergio Ríos	Se logró implementar las primeras versiones de las funciones de candidatos y selección del algoritmo codicioso
20/09/2020	Julián Camacho José Fabián Mendoza Mata	Se realizaron funciones para verificar si existe un ganador o si se generó un empate. Para este segundo caso se implementó la funcionalidad de verificar si una columna o la matriz están llenas. Para el caso de que exista un ganador, se desarrollaron funciones para revisar si hay 4 fichas en línea recta, ya sea en una fila o una columna. Para las diagonales fue necesario investigar en la página web del primer punto de la bibliografía. Con ello, se finaliza la parte mayoritaria de la lógica del juego.
20/09/2020	Julián Camacho Fabián Mendoza	Reunión para integrar la lógica del juego con la interfaz hasta el momento. Se consultó la página web del segundo punto de la bibliografía para importar funciones de un diferente archivo Racket.

21/09/2020	Sergio Ríos	Segunda versión de funciones candidatos y selección del algoritmo codicioso
22/09/2020	Todos	Reunión para informar sobre las funcionalidades implementadas hasta el momento, parte de la lógica del juego, versión inicial de la interfaz y adelanto del algoritmo codicioso.
22/09/2020	Sergio Ríos	Primera versión de funciones viabilidad, objetivo y solución del algoritmo codicioso
25/09/2020	Todos	Reunión para visualizar avances en el desarrollo y resolver consultas en la implementación del algoritmo codicioso.
25/09/2020	Sergio Ríos	Corrección de errores en funciones de viabilidad y objetivo del algoritmo codicioso
25/09/2020	Sergio Ríos José Fabián Mendoza	Implementación del algoritmo codicioso a la interfaz del juego
29/09/2020	Sergio Ríos	Corrección de errores en el algoritmo codicioso