

Documentación Tarea Programada 1.

Lenguajes compiladores e intérpretes.

Paradigma Funcional.

“four\_in\_line\_racket”

Profesor:

Marco Rivera Meneses.

Integrantes:

Carlos Andrés Mata Calderón

David Robles

Victor Cruz Jiménez.

Grupo 4.

Fecha de entrega:



# Descripción del problema.

La siguiente tarea programada tiene como fin el realizar el juego clásico de “4 en línea” utilizando el paradigma funcional de Racket.

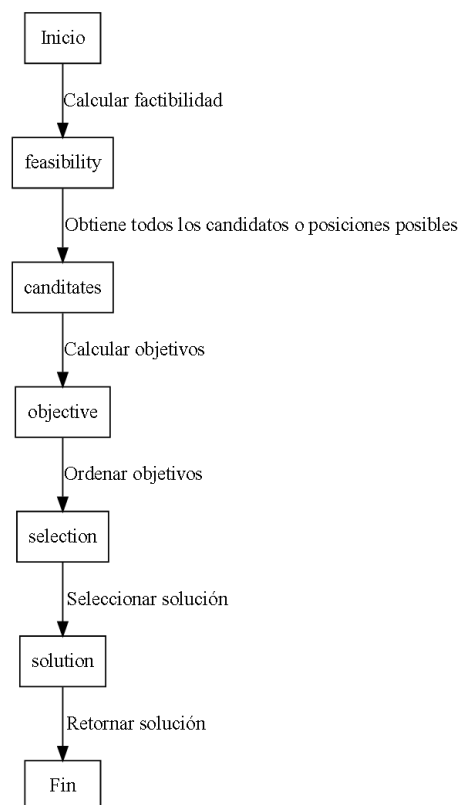
Enunciado de la tarea:

“El 4 en Línea pertenece a la familia de juegos populares ya que sus sencillas reglas se pueden aprender rápidamente ofreciendo un gran entretenimiento durante un largo tiempo. El objetivo de este juego consiste en colocar cuatro fichas en una fila continua vertical, horizontal o diagonalmente. Se juega sobre un tablero de 8x8 casillas que al empezar está vacío.”

Cabe aclarar que el tamaño del tablero puede ser desde un 8x8 hasta un 16x16, esto queda a elección del usuario

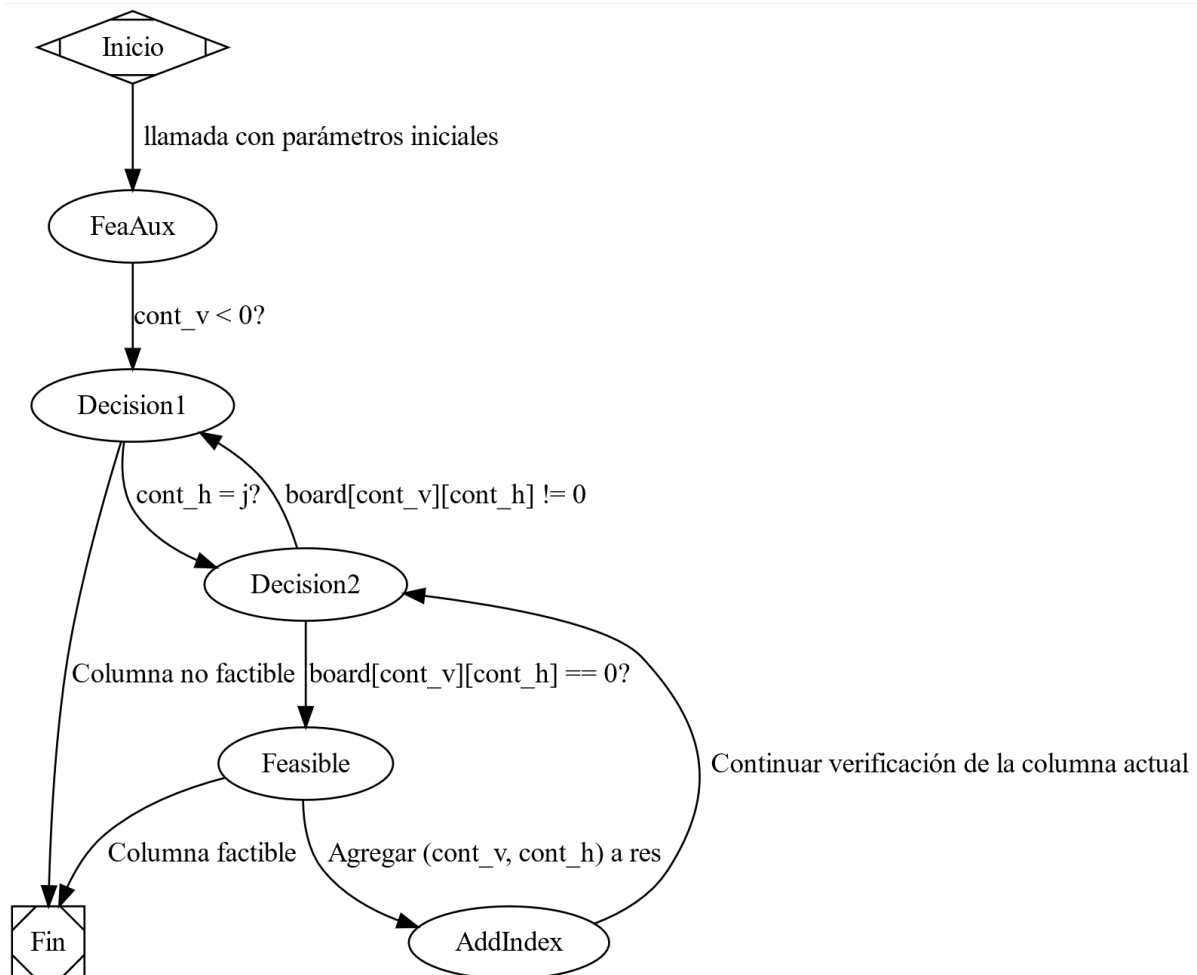
# Descripción de algoritmos utilizados.

**Función “greedy”.** Esta función representa una estrategia de juego llamada "Greedy", Consiste en seleccionar la opción que maximice el puntaje del jugador en cada jugada, sin considerar el futuro a largo plazo. Recibe el tablero de juego, player, que corresponde al valor del jugador (varía entre 1 y 2) y enemy, el valor del jugador enemigo (1-2). Devuelve una solución óptima para el jugador actual.



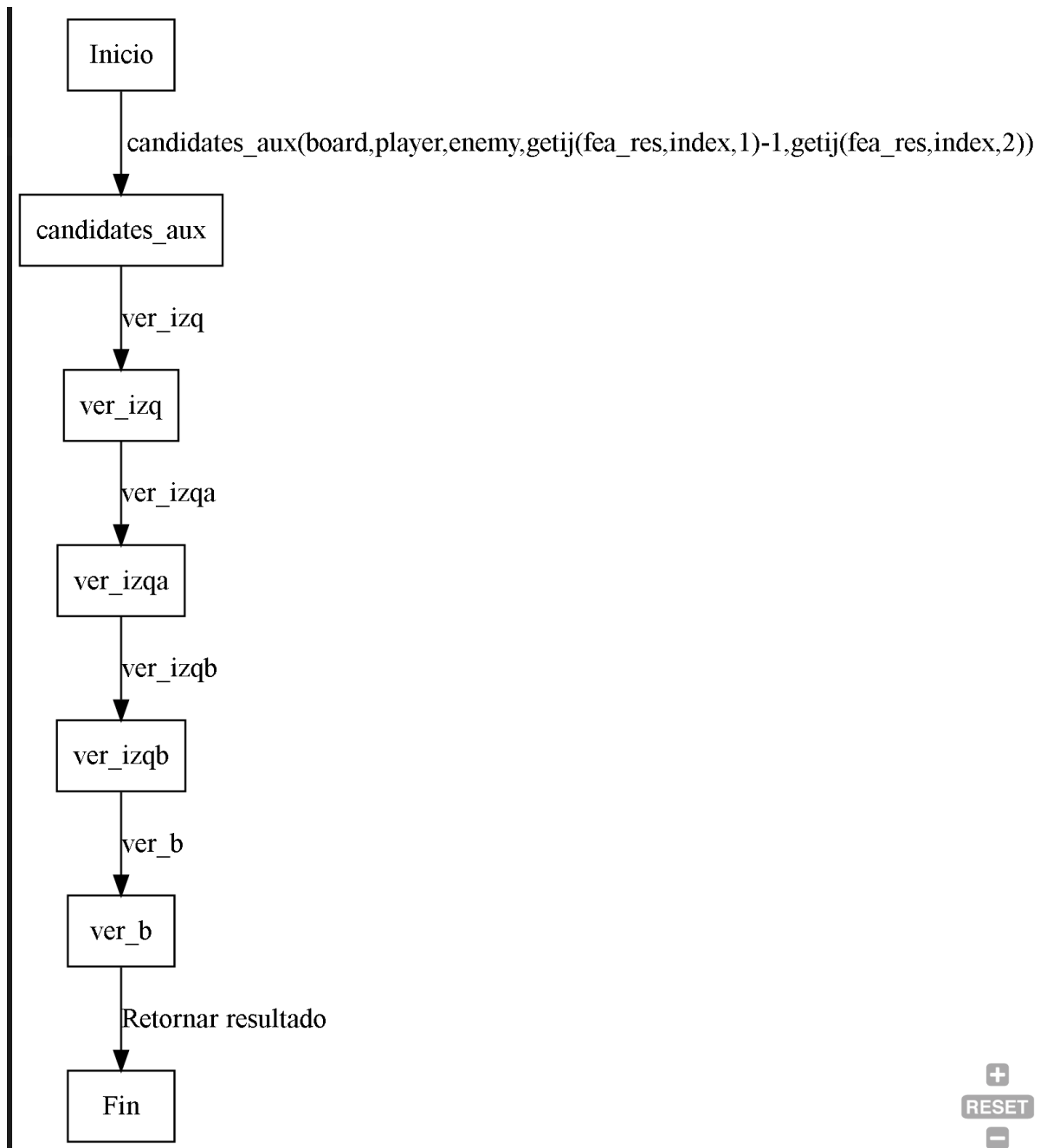
Se comienza verificando la factibilidad de las posibles jugadas en el tablero, luego se obtienen todas las posiciones posibles, se calculan los objetivos, se ordenan los objetivos y se selecciona la solución. Finalmente se retorna la solución. El diagrama muestra de manera visual las decisiones que se toman en cada paso del proceso.

**Función “feasibility”.** Esta función verifica la factibilidad de las posibles jugadas en el tablero, es decir, si una columna aún tiene espacios disponibles para colocar fichas.



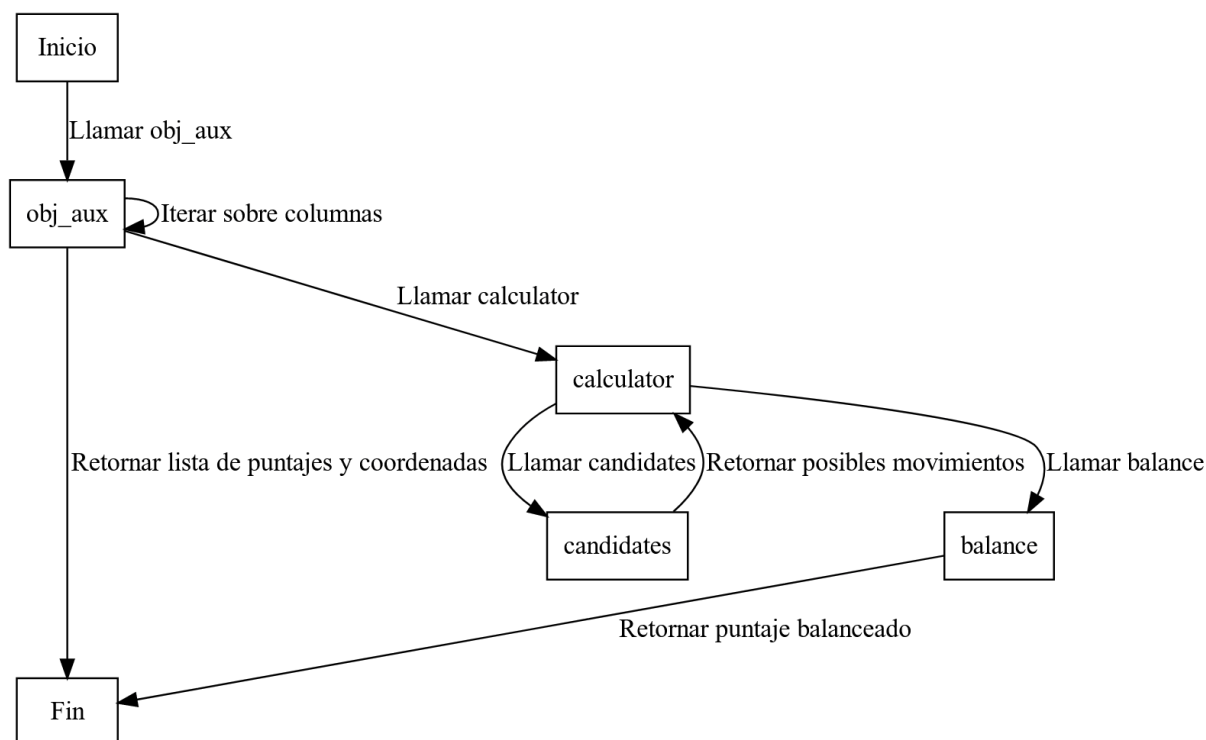
La función se llama a "fea\_aux" con los parámetros iniciales. "fea\_aux" verifica si la posición actual en el tablero tiene un valor de 0 (significando que está disponible para colocar una ficha) o no. Si es 0, agrega los índices de fila y columna a la lista "res" y continúa verificando la columna actual. Si no es 0, continúa verificando la columna actual. Si ha terminado de verificar toda la columna sin encontrar una posición disponible, la columna no es factible y se finaliza la función. Si se ha encontrado una posición disponible, la columna es factible y se finaliza la función. La decisión de seguir verificando la columna actual o pasar a la siguiente columna depende de si la columna actual tiene aún filas por verificar ( $\text{cont\_v} \geq 0$ ) y si se ha llegado al final de la columna actual ( $\text{cont\_h} = j$ ).

**Función “candidates”** . Esta función se utiliza para obtener los posibles movimientos o candidatos para una jugada en el tablero en la columna especificada.



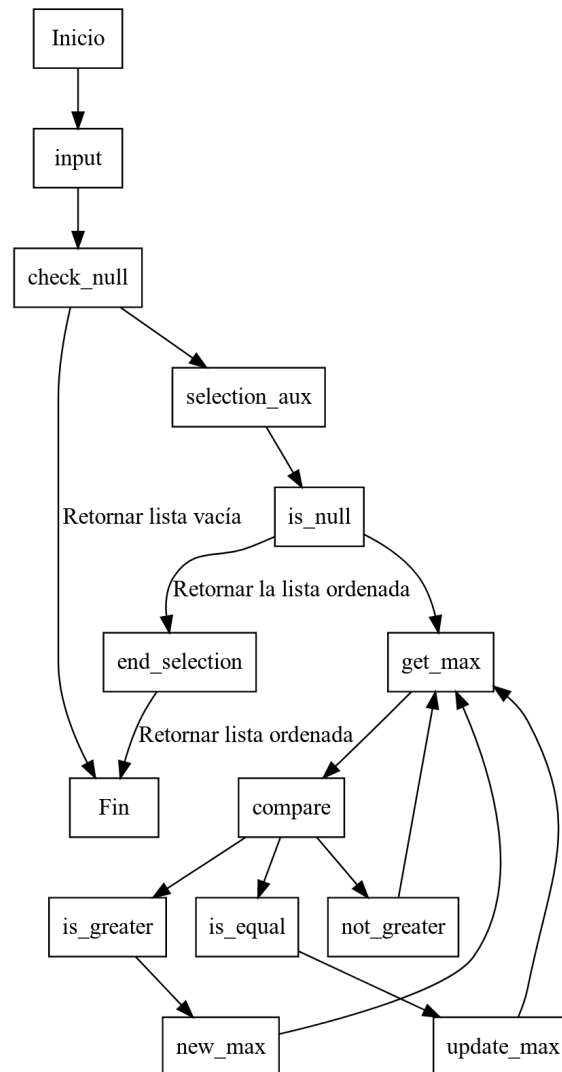
La función llama a la función auxiliar `candidates_aux`, que a su vez llama a diferentes funciones para verificar las posibilidades de movimientos válidos en diferentes direcciones (izquierda, arriba-izquierda, abajo-izquierda y abajo) para una columna específica en el tablero. Si se encuentra un movimiento válido, se agrega a una lista de posibles candidatos. Si no se encuentra ningún movimiento válido, la función devuelve 0. Finalmente, la lista de posibles movimientos se devuelve a la función `greedy` para que se pueda calcular el objetivo y seleccionar la solución adecuada.

**Función “objective”.** Esta función se encarga de calcular el objetivo del jugador actual, es decir, el puntaje total que se puede utilizar.



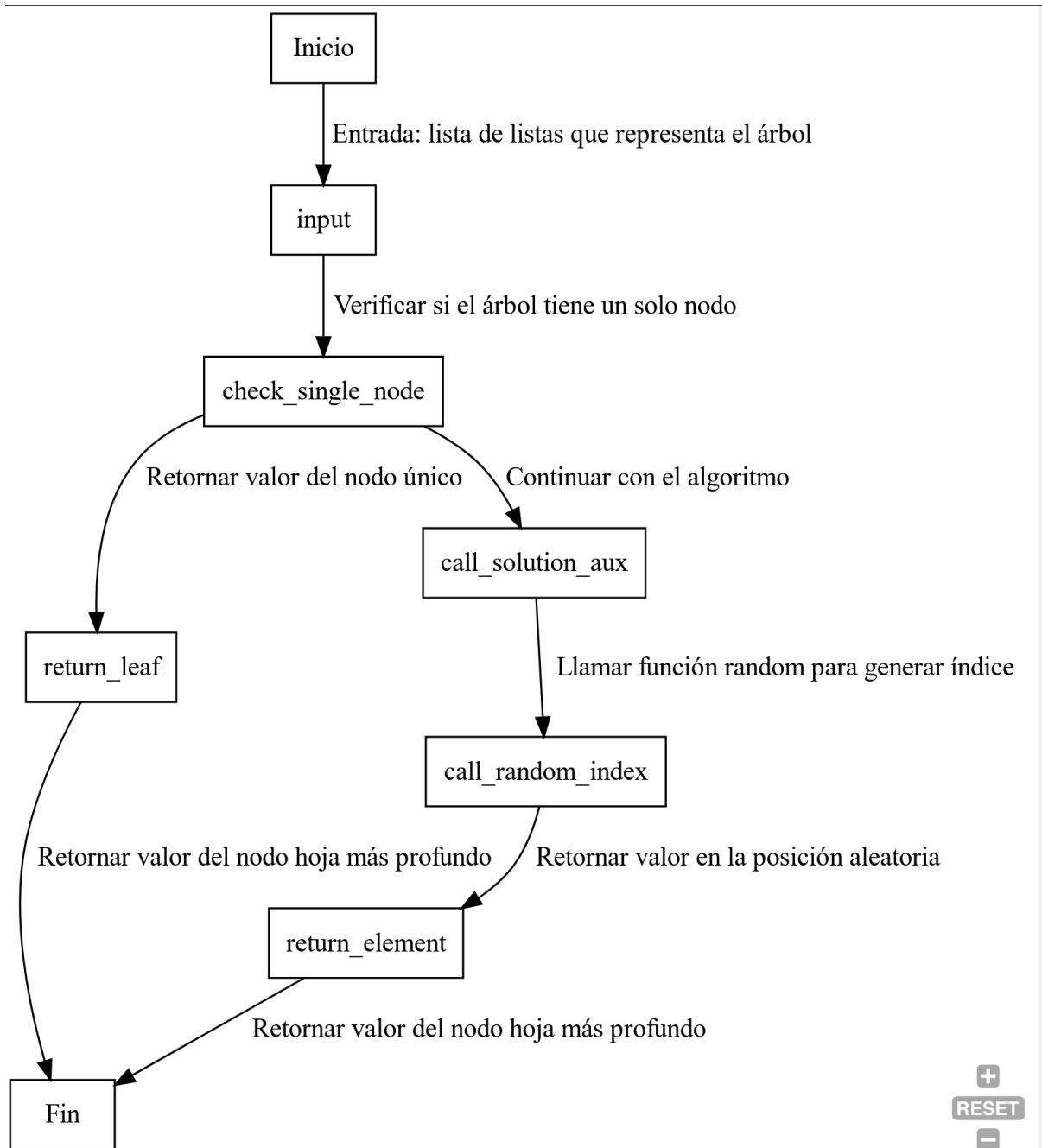
El diagrama de flujo representa el proceso de cálculo del objetivo del jugador actual en el juego de conecta 4. Comienza con la función "objective", que llama a la función auxiliar "obj\_aux". Esta última función utiliza la función "calculator" para calcular el puntaje de cada posible jugada en una columna dada y la función "candidates" para obtener los posibles movimientos en esa columna. Luego, se utiliza la función "balance" para balancear los puntajes obtenidos por el jugador actual y el jugador enemigo, tomando en cuenta la posibilidad de conseguir una posición ganadora o evitar una posición perdedora. La función "obj\_aux" utiliza recursion para iterar sobre todas las columnas con espacios disponibles, almacenando en una lista el puntaje y las coordenadas de cada posible jugada. Al final, la función "objective" retorna una lista con los puntajes y coordenadas de cada posible jugada en cada columna del tablero.

**Función "selection".** Esta función implementa el algoritmo de ordenamiento Selection Sort para ordenar una lista de puntos en orden descendente, de manera que la primera posición de la lista tenga el puntaje más alto.



El diagrama de flujo representa el algoritmo de ordenamiento Selection Sort. El proceso comienza con la entrada de una lista de puntos que se desea ordenar en orden descendente por su puntaje. A continuación, se define una función auxiliar llamada "selection\_aux" que utiliza la recursión para iterar sobre la lista y comparar el puntaje de cada elemento con el puntaje del elemento máximo actual. Si se encuentra un elemento con un puntaje más alto que el actual máximo, se actualiza el máximo para incluir sólo ese elemento. Si se encuentra un elemento con el mismo puntaje que el máximo actual, se agrega a la lista de máximos. Si el puntaje del elemento es menor que el máximo actual, se ignora y se continúa con el siguiente elemento. Una vez que se ha recorrido toda la lista, se devuelve la lista de máximos ordenada en orden descendente.

**Función “solution”.** La función toma una lista de listas, donde cada sublista representa un elemento. La función devuelve el valor almacenado en el último elemento de la lista.



El diagrama de flujo comienza con el nodo "start" y luego se mueve al nodo "input", que representa la lista de listas de entrada. A partir de ahí, el diagrama de flujo verifica si la lista de entrada tiene solo un nodo o no. Si hay solo un nodo, se devuelve el valor del único nodo y el diagrama de flujo termina. De lo contrario, el diagrama de flujo llama a la función "solution\_aux" para encontrar el valor con la profundidad más profunda. La función "solution\_aux" primero llama a la función "random" para generar un índice para un elemento aleatorio en la lista de entrada y luego devuelve el valor en ese índice. Finalmente, el diagrama de flujo devuelve el valor del nodo hoja con la profundidad más profunda, que es la salida de la función "solution".

# Descripción de funciones utilizadas.

**Función “create-matrix”.** La función anterior tiene como parámetros de entrada valores de  $n$  y  $m$ , los cuales son valores de números enteros, los cuales establecen las dimensiones de la matriz a utilizar, a su vez, llenan dicha matriz de 0.

**Función “print-matrix”.** Dicha función recibe una matriz, la cual se imprime en consola, esta función se utiliza únicamente para verificar que todo avance de manera esperada.

**Función “insert-token”.** Esta función inserta un token en la columna dada en la matriz, de acuerdo con la regla de un juego de 4 en línea. El token se inserta en la fila más baja disponible. Tiene como valores de entrada.  $col$ =: la columna en la que se desea insertar el token,  $board$ : la matriz que representa el tablero de juego,  $player$ : el valor del jugador que está insertando el token. Por último, retorna la matriz de juego actualizada.

**Función “get-ele-matrix”.** Esta función retorna el elemento en la posición  $(i, j)$  de la matriz. Recibe como parámetros  $i$ = la fila,  $j$ = la columna de la matriz.

**Función “replace-ele-matrix”.** Reemplaza un valor en una matriz. Recibe  $i$ =fila,  $j$ = columna,  $board$ = matriz a cambiar,  $val$ = valor que se quiere colocar en la posición  $(i,j)$ . Retorna la matriz con el valor reemplazado.

**Función “replace-ele-list”.** Función que reemplaza un elemento en una lista. Recibe:  $por$ = el índice del elemento a reemplazar,  $list$ = la lista en la cual se quiere reemplazar el elemento,  $var$ = el valor que se quiere colocar. Retorna la lista modificada con el valor reemplazado.

**Función “check-tie”.** Función encargada de verificar si hay un empate en el tablero, recibe como parámetro de entrada una matriz, devuelve un  $\#t$  si no hay celdas vacías en el tablero, o retorna  $\#f$  en cualquier otro caso.

**Función “check-win”.** Función encargada de verificar si se cumple la condición de victoria del juego, recibe una matriz que representa el tablero de juego, y el número que representa al jugador. Retorna  $\#t$  si hay una secuencia de cuatro elementos iguales del jugador en una fila, columna o diagonal, en caso de que no se cumpla retorna  $\#f$ .

**Función “check-consecutive”.** Función encargada de verificar si hay un número consecutivo de elementos en una lista, recibe como parámetros de entrada una lista y el elemento que se busca contar, retorna  $\#t$  si hay al menos 4 elementos consecutivos iguales en la lista, retorna  $\#f$  en cualquier otro caso.

**Función “get-left-diagonal”.** Obtiene los elementos de la diagonal izquierda que pasa por la posición  $(row, col)$  de una matriz dada, recibe  $row$  índice de fila,  $col$  índice de columna,  $board$  la matriz de juego, retorna una lista con los elementos en la diagonal izquierda secundaria.



**Función “get-right-diagonal”.** Obtiene los elementos de la diagonal derecha secundaria de una matriz, recibe row índice fila, col índice columna, board matriz de juego, retorna una lista con los elementos en la diagonal derecha secundaria.

**Función “get-col”.** Función encargada de obtener los elementos de una columna específica del tablero, tiene como valores de entrada col índice de columna y board, la cual es la matriz de juego, retorna una lista con los elementos de la columna seleccionada.

**Función “get-row”.** Función encargada de obtener los elementos de una fila específica del tablero, tiene como valores de entrada row índice de fila y board, la cual es la matriz de juego, retorna una lista con los elementos de la fila seleccionada.

## Descripción de estructuras utilizadas.

La principal estructura utilizada para realizar de manera óptima la presente tarea fueron las listas y matrices, entiéndase por listas un conjunto de datos agrupados, dichos datos deben de ser del mismo tipo, a su vez, cada dato tiene una posición específica en la lista.

Las listas se utilizaron para formar una matriz, la cual en palabras simples se puede entender como “una lista de listas”, es decir, una matriz en programación se basa en agrupar un número de listas, las cuales tienen las mismas dimensiones.

Dicha matriz se utilizó con el fin de representar de forma lógica las dimensiones de juego y a su vez, en dicha matriz se realizaron todas las comprobaciones lógicas para verificar el correcto funcionamiento del juego.

Ahora bien, cabe recalcar que el funcionamiento del juego se optó por crear una matriz de botones para representar de forma visual el tablero de juego, a su vez, de este modo se puede tener de manera exacta la posición en la que se desea colocar una ficha.

## Problemas sin solución.

**Pintar de colores los botones.** Se encontró un problema al intentar pintar los botones de colores en la GUI del juego. Se descubrió que no existe una forma de cambiar el color de los botones en Racket, lo cual limitaba la posibilidad de asignar un color específico a cada jugador (por ejemplo, rojo para el jugador 1 y azul para el jugador 2). Ante la imposibilidad de implementar esta característica, se decidió eliminarla del diseño del juego.

## Problemas encontrados.

**Diagonal secundaria izquierda.** La función get-left-diagonal tenía una condición que evaluaba si la fila row era mayor o igual a la longitud de la matriz board, o si la columna col era menor que 0. Si alguna de estas condiciones se cumplía, la función devolvía la lista acc, que en ese punto solo contenía los elementos de la fila row. Este problema se corrigió cambiando la primera condición del condicional para verificar si la columna col era menor

que 0 y si la fila row estaba dentro del rango válido. Otro problema que se encontró fue que el algoritmo no manejaba correctamente matrices rectangulares, ya que la línea (cons (list-ref (list-ref board row) col) acc) podía producir un error de índice si la fila row y la columna col estaban fuera del rango de índices para la matriz board. Para solucionarlo, se modificó el caso base del algoritmo para manejar este caso, permitiendo que la función pudiera trabajar con matrices de cualquier tamaño o forma. Estos cambios permitieron que la función get-left-diagonal pudiera obtener correctamente los elementos de la diagonal izquierda secundaria para cualquier matriz, independientemente de su forma o tamaño.

**Chequear consecutivo 4 en línea.** El código presentado tenía un problema en la condición del primer caso del condicional solo se evaluaba cuando se habían recorrido todas las filas de la matriz. Esto causaba errores de índice al llamar a la función auxiliar check-all-aux con valores no válidos de i y j. Para solucionar este problema, se modificó la condición del primer caso del condicional cond para verificar si tanto i como j eran mayores o iguales a 0 y si estaban dentro del rango válido para la matriz board. Con este cambio, la función check-all ahora puede manejar correctamente matrices de cualquier tamaño y forma, sin producir errores de índice. En resumen, se encontró un error de índice en el código original que se soluciona modificando la condición del condicional para verificar si los índices estaban dentro del rango válido de la matriz.

**Entrada de datos de la GUI.** Se presentó un problema en la entrada de datos de la GUI del usuario. Se requería que los usuarios ingresaran números para la cantidad de filas y columnas, y se necesitaba comprobar que los datos fueran correctos. La condición necesaria para asegurarse de que los datos fueran válidos requería demasiado esfuerzo para implementarse, por lo que se decidió utilizar una alternativa similar. Se consideró el uso de combobox, pero Racket no tiene este widget, así que se buscó una alternativa y se encontró el slider. Este es un widget que permite establecer un rango de valores para recibir datos. Esta solución resultó muy útil ya que no solo eliminó la necesidad de comprobar si la entrada de texto era un número válido, sino que también permitió establecer de una vez que las filas y columnas debían ser valores enteros entre 8 y 16. Además, se agregó la opción de que los jugadores pudieran elegir si querían ser la ficha 1 o la ficha 2 utilizando este mismo método.

**Botones del GUI.** Se presentaron varios errores al intentar crear el grid con los botones para la GUI del juego. El primer problema fue que, al principio, no se estaban guardando los botones creados en ningún lado, por lo que no se podía hacer referencia a ellos al actualizar los botones más tarde. Para solucionar esto, se creó una matriz igual a la matriz board utilizada en la lógica del juego y se guardaron los botones en ella. Por último, hubo otro error en la construcción del grid. Cuando se presionaba un botón, el valor actualizado de la matriz de juego se establecía en las primeras filas en lugar de las últimas, como se esperaba según la lógica del juego. Se descubrió que esto se debía a que se estaba utilizando la función cons, que agrega elementos al principio de la lista, en lugar de append, que agrega elementos al final. Al cambiar a append, se resolvió el problema.

**Insertar ficha.** Se encontró un problema en la función insert-token del juego, ya que cuando se insertaba una nueva ficha en la matriz de juego, esta no se apilaba correctamente, sino que simplemente se sobrescribe en la misma posición. Se revisó el código y se encontró que lo que faltaba eran funciones auxiliares que ayudarán a obtener un elemento de la matriz y establecer elementos en la misma. Además, se descubrió que la lógica de buscar

un espacio vacío en la columna para colocar la ficha requería una recursión que empezaba desde la fila 0 hasta la fila final (por ejemplo, 16), lo cual era ineficiente. Por lo tanto, se modificó la lógica para que se comenzará desde la fila de abajo hacia arriba (es decir, empezando desde la fila 16) y se agregó una condición que devolvía la matriz de juego original si la columna ya estaba llena de fichas. Esto mejoró la eficiencia y aseguró que las fichas se apilaran correctamente en la matriz de juego.

## Plan de actividades.

Estudiante a cargo:	Tarea asignada:	Descripción de tarea:	Tiempo estimado:	Fecha de entrega:
Carlos Andrés Mata Calderón	Lógica de juego	Realizar de manera óptima la lógica del juego propuesto.	10 Horas	9/3/2023
	Gui	Mediante bibliotecas gráficas, realizar un tablero de juego	4 Horas	9/3/2023
David Robles	Algoritmo Voraz	Encargado de la realización del algoritmo voraz	10 Horas	9/3/2023
Victor Cruz Jiménez	Gui	Mediante bibliotecas gráficas, realizar una GUI para el usuario en la cual se muestran todos los elementos necesarios	4 Horas	9/3/2023
	Administración	Se encargará de la realización de la documentación necesaria para el correcto entendimiento de la tarea	3 Horas	9/3/2023

## Conclusiones.

Una vez realizada la tarea establecida, se llegó a la conclusión de que gracias a la correcta implementación del paradigma funcional, es posible realizar funciones y algoritmos en una menor cantidad de líneas de código, pero el hecho de que sea una forma de programación que se utiliza en menor cantidad a diferencia de otros paradigmas, implica que se tenga que hacer un mayor tiempo de planeación a la hora de encarar diversos problemas.

Otro punto que es importante recalcar, es que gracias al paradigma funcional es posible realizar programas más eficientes, ya que al evitar el uso de variables se necesita una menor cantidad de memoria por lo que la velocidad de ejecución del código va a ser más veloz, a diferencia de que si se usarán variables.

## Recomendaciones.

Como principal recomendación a la hora de realizar un programa con el paradigma funcional, sería de que se investigue con antelación acerca de la correcta implementación del paradigma, para de este modo evitar gran cantidad de trabas en la elaboración del programa planteado.

Ya que una de las principales características del paradigma funcional es evitar el uso de variables, la sección lógica del programa debería evitar el uso de las mismas, pero en cambio, la implementación de algunas variables para guardar datos importantes en la GUI es una opción viable y más que recomendada a la hora de realizar un proyecto en racket.

# Referencias.

Racket. (s.f.). Racket Documentation. Recuperado de <https://docs.racket-lang.org/>

Racket. (s.f.). GUI Documentation. Recuperado de <https://docs.racket-lang.org/gui/>

Racket. (s.f.). Creating Windows. En GUI Documentation. Recuperado de [https://docs.racket-lang.org/gui/windowing-overview.html#%28part.\\_.Creating\\_.Windows%29](https://docs.racket-lang.org/gui/windowing-overview.html#%28part._.Creating_.Windows%29)

Estes, M. (2013, 6 de diciembre). C++ Example Code [Gist]. GitHub. <https://gist.github.com/MichaelEstes/7836988>

Wikipedia. (s.f.). Algoritmo voraz. En Wikipedia, la enciclopedia libre. Recuperado de [https://es.wikipedia.org/wiki/Algoritmo\\_voraz](https://es.wikipedia.org/wiki/Algoritmo_voraz)

GeeksforGeeks. (s.f.). Greedy Algorithms. Recuperado de <https://www.geeksforgeeks.org/greedy-algorithms/>

# Bitácora.

## Carlos.

- **01/03/2023: Duración: De 10:00 a.m. a 6:00 p.m.** Se trabajó en la lógica del juego y se crearon varias funciones auxiliares para manejar la matriz y los botones del tablero. Se crearon funciones para obtener elementos de una matriz y establecer elementos de una matriz.
- **02/03/2023: Duración: De 9:00 a.m. a 7:00 p.m.** Se continuó trabajando en la lógica del juego y se solucionaron problemas con la inserción de fichas en la board y la verificación de si hay un ganador. También se realizaron mejoras en las pruebas unitarias y se eliminó la función de cambiar el color de los botones debido a la falta de una forma adecuada de hacerlo en Racket.
- **05/03/2022: Duración: De 11:00 a.m. a 8:00 p.m.** Se trabajó en la creación del grid de botones para representar el tablero del juego. Se solucionaron problemas relacionados con la identificación de la posición de un botón en la matriz y la actualización de los botones con los nuevos valores de la board.
- **06/10/2022: Duración: De 5:00 p.m. a 8:00 p.m.** Se finalizó la ventana de la GUI.
- **07/10/2022: Duración: De 3:00 p.m. a 5:00 p.m.** Se corrigieron errores de ortografía en el código.

## David.

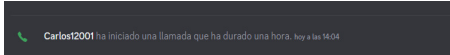
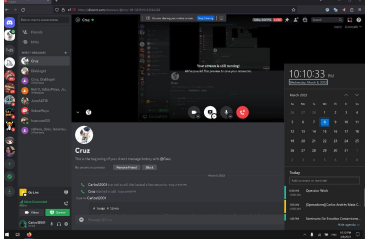
- **01/03/2023: Duración: De 1 p.m. a 3:30 p.m.** Comencé diseñando en papel la lógica de una función que recorriera una matriz o tablero de forma horizontal hacia la izquierda.

- **02/03/2023: Duración: De 7:30 p.m. a 11:00 p.m.** Escribí la función en código que obtiene un puntaje horizontalmente, di comienzo a las funciones diagonales utilizando una lógica similar.
- **05/03/2023: Duración: De 4:00 p.m. a 7:00 p.m.** Finalicé las funciones que obtienen puntos alrededor de una posición específica de la matriz o tablero. Dejé escrito de manera textual y lógica las funciones necesarias para concluir el algoritmo codicioso.
- **07/03/2023: Duración: De 7:30 p.m. a 12:00 a.m.** Trabajé en una función que hiciera uso de las funciones para calcular puntajes de manera que agregara cada puntaje a una coordenada, tomando en cuenta el puntaje que obtiene el jugador contrario en la misma coordenada.
- **09/03/2023: Duración: De 8:30 p.m. a 11:00 p.m.** Le di formato a las funciones creadas anteriormente para así seguir y finalizar las instrucciones dadas en la sección del algoritmo codicioso, fue necesario crear algunas pequeñas funciones o auxiliares simples para lograrlo.

#### **Victor.**

- **03/03/2023: Duración: De 9:00 p.m. a 11:30 p.m.** Se comenzó con una implementación simple de la GUI, para tener una guía de cómo llevar a cabo la correcta elaboración de la misma.
- **03/05/2023: Duración: De 3:00 p.m. a 6:30 p.m.** Se continuó con el desarrollo de la interfaz gráfica. Se realizó lo necesario para crear un tablero de juego claro para el usuario. Investigación acerca de cómo solucionar errores a la hora de la implementación de una matriz de botones.
- **03/06/2023: Duración: De 5:00 p.m. a 8:00 p.m.** Se finalizó la gui. A su vez se realizó una reunión para aclarar dudas acerca de la sección de lógica del juego.

**Minutas de reuniones:**

Fecha	Asistentes	Descripción	Evidencia
28/02/2023	Carlos, David, Victor	Creación del grupo de WhatsApp y asignación de tareas. Carlos se encargará de la lógica del juego, David del algoritmo Greedy y Víctor de la creación de la interfaz gráfica de usuario (GUI).	-Forma Presencial-
02/03/2023	Carlos, David	Se discutió cómo funcionará la lógica del juego y el algoritmo Greedy. Se acordó que el algoritmo Greedy devolverá la columna óptima en la que colocar la ficha, y recibirá como entrada la board actual del juego, el token del jugador actual y el token del jugador enemigo.	
05/03/2022	Carlos, Victor	En resumen, se acordó que la GUI debe representar la matriz del juego mediante un grid de botones y que Victor se encargará de la documentación y el manual de usuario.	

07/10/2022

Carlos, David

Ayuda para la unificación entre el juego y el greddy. Se acordó que Carlos se encargaría de implementar esta comunicación entre el juego y Gredy.

