



**Área de Ingeniería en Computadores**

**Lenguajes, Compiladores e Intérpretes**

**I Tarea Programada:**

**4Line**

**Profesor**

Marco Rivera Menénes

**Estudiantes**

Gabriel Chacón Alfaro - 2021049454 [gchacon8@estudiantec.cr](mailto:gchacon8@estudiantec.cr)

Emanuel Marín Gutiérrez 2019067500 [emarin702740530@estudiantec.cr](mailto:emarin702740530@estudiantec.cr)

Jose Andres Rodriguez Rojas 2019279722 [joseandres216@estudiantec.cr](mailto:joseandres216@estudiantec.cr)

**Fecha**

**11 de marzo, 2023**

## **Contenido**

- 1.1 Descripción de los algoritmos desarrollados
- 1.2 Descripción de las funciones implementadas
- 1.3 Descripción de las estructuras de datos utilizadas
- 1.4 Problemas sin solución
- 1.5 Problemas encontrados
- 1.6 Plan de actividades
- 1.7 Conclusiones
- 1.8 Recomendaciones
- 1.9 Bibliografía
- 1.10 Bitácora

## 1.1 Descripción de los algoritmos de solución desarrollados

### Algoritmo Goloso (Greedy Algorithm)

Se desarrolló un algoritmo goloso con la intención de permitir a la computadora realizar sus movimientos una vez que llegue su turno. Este algoritmo se basa en un proceso de cinco etapas que, en conjunto, permiten a la computadora revisar, evaluar y escoger la posición más óptima para jugar en su turno.

- a) Función candidatos: En el caso de la función candidatos, esta se encargaría de revisar y seleccionar todos los espacios que se encuentran vacíos en la matriz (tablero de juego), para efectos de este programa, se decidió seguir la siguiente simbología a la hora de trabajar lógicamente la matriz:

0: Representa todo espacio en el que no hay ninguna ficha.

1 o 2: Representan los espacios donde ya haya alguna ficha del jugador o de la computadora (Depende del color de ficha que seleccione el jugador).

Sin embargo, por recomendación del profesor Marco Rivera Meneses, se optó por omitir el paso de la función candidatos, ya que al momento de realizar la consulta, se nos indicó que la función que el equipo ya había codificado, realizaba satisfactoriamente el trabajo que las funciones candidatos y viabilidad debían sobrellevar por separado.

- b) Función viabilidad: Esta función se hace cargo de seleccionar los candidatos reales en los que se puede ubicar una ficha (teniendo en cuenta que esta cae hasta el fondo o hasta encontrar el último espacio vacío antes de uno que ya esté previamente ocupado). En este caso, esta función toma todos esos candidatos reales y crea un par ordenado para cada uno, según el formato:

*(columna, fila)*

De este modo, esta función realiza un llamado a la función objetivo y le provee la lista de candidatos reales utilizando el estándar previamente establecido.

- c) Función objetivo: Esta corresponde a una de las funciones más importantes dentro del proceso del algoritmo goloso, ya que es la responsable de asignar un valor a cada uno de los candidatos reales que provienen de la función viabilidad.

Para efectos de este programa, se decidió establecer la siguiente estructura de puntos, según los siguientes escenarios:

- I) Función connecting4 (6 puntos): La función connecting4 verifica si, en la posición dada, la computadora conectaría cuatro fichas en fila (ganaría la partida), está es la función mejor valorada ya que se busca que el objetivo de la computadora sea ganar en lugar de conectar 3 fichas en fila o inclusive bloquear una posible victoria del jugador.

```
(define (connecting4 matrix column row playerUsingIA)
  (cond ((= (+ (vertical-win-down column row matrix 0 playerUsingIA #t) (vertical-win-up column row matrix 0
playerUsingIA #t) 1) 4) #t)
        ((= (+ (horizontal-win-down column row matrix 0 playerUsingIA #t) (horizontal-win-up column row matrix 0
playerUsingIA #t) 1) 4) #t)
        ((= (+ (pair-diagonal-win-down column row matrix 0 playerUsingIA #t) (pair-diagonal-win-up column row matrix
0 playerUsingIA #t) 1) 4) #t)
        ((= (+ (odd-diagonal-win-down column row matrix 0 playerUsingIA #t) (odd-diagonal-win-up column row matrix 0
playerUsingIA #t) 1) 4) #t)
        (else #f)))
```

- II) Función blocking-rivals-win (5 puntos): En este caso, la función se revisará si, en el candidato a evaluar, se está bloqueado una posible victoria del jugador, asignando cinco puntos al candidato para que así, la computadora sea capaz de competir ante el jugador y no permitirle ganar tan fácilmente.

```
(define (blocking-rivals-win matrix column row playerUsingIA)
  (cond ((= playerUsingIA 1) (blocking-rival-win-aux matrix column row 2))
        (else (blocking-rival-win-aux matrix column row 1))))

(define (blocking-rival-win-aux matrix column row playerUsingRival)
  (cond ((= (+ (vertical-win-down column row matrix 0 playerUsingRival #t) (vertical-win-up column row matrix 0
playerUsingRival #t) 1) 4) #t)
        ((= (+ (horizontal-win-down column row matrix 0 playerUsingRival #t) (horizontal-win-up column row matrix 0
playerUsingRival #t) 1) 4) #t)
        ((= (+ (pair-diagonal-win-down column row matrix 0 playerUsingRival #t) (pair-diagonal-win-up column row
matrix 0 playerUsingRival #t) 1) 4) #t)
        ((= (+ (odd-diagonal-win-down column row matrix 0 playerUsingRival #t) (odd-diagonal-win-up column row matrix
0 playerUsingRival #t) 1) 4) #t)
        (else #f)))
```

- III) Función blocking-rivals-movement (4 puntos): Está función posee un comportamiento muy similar al de la función blocking-rivals-win, sin embargo, se le asignan menos puntos ya que este movimiento no es determinante en el resultado de la partida (o al menos no de forma inmediata). Este paso evalúa si el candidato interrumpe una potencial fila de tres fichas por parte del jugador, esto con el fin de brindar dinamismo a

los movimientos de la computadora y, a su vez, evitar que el jugador ponga a la computadora en una situación donde perdería inevitablemente.

```
(define (blocking-rivals-movement matrix column row playerUsingIA)
  (cond ((= playerUsingIA 1) (blocking-rival-movement-aux matrix column row 2))
        (else (blocking-rival-movement-aux matrix column row 1))))

(define (blocking-rival-movement-aux matrix column row playerUsingRival)
  (cond ((= (+ (vertical-win-down column row matrix 0 playerUsingRival #t) (vertical-win-up column row matrix 0
playerUsingRival #t) 1) 3) #t)
        ((= (+ (horizontal-win-down column row matrix 0 playerUsingRival #t) (horizontal-win-up column row matrix 0
playerUsingRival #t) 1) 3) #t)
        ((= (+ (pair-diagonal-win-down column row matrix 0 playerUsingRival #t) (pair-diagonal-win-up column row
matrix 0 playerUsingRival #t) 1) 3) #t)
        ((= (+ (odd-diagonal-win-down column row matrix 0 playerUsingRival #t) (odd-diagonal-win-up column row matrix
0 playerUsingRival #t) 1) 3) #t)
        (else #f)))
```

- IV) Funciones `connecting3`, `connecting2` y `connecting1` (3 , 2 y 1 punto respectivamente): Estas funciones son muy similares entre ellas y poseen el mismo objetivo, es decir, ayudar a que la computadora realice movimientos que le permitan conectar una, dos o incluso tres fichas en fila en el candidato que se evalúa en ese momento. De este modo la computadora crea paulatinamente posibles escenarios donde le resulte más fácil ganar la partida.

```
(define (connecting3 matrix column row playerUsingIA)
  (cond ((= (+ (vertical-win-down column row matrix 0 playerUsingIA #t) (vertical-win-up column row matrix 0
playerUsingIA #t) 1) 3) #t)
        ((= (+ (horizontal-win-down column row matrix 0 playerUsingIA #t) (horizontal-win-up column row matrix 0
playerUsingIA #t) 1) 3) #t)
        ((= (+ (pair-diagonal-win-down column row matrix 0 playerUsingIA #t) (pair-diagonal-win-up column row matrix 0
playerUsingIA #t) 1) 3) #t)
        ((= (+ (odd-diagonal-win-down column row matrix 0 playerUsingIA #t) (odd-diagonal-win-up column row matrix 0
playerUsingIA #t) 1) 3) #t)
        (else #f)))
```

A la hora de asignar los puntos a los candidatos se agrega un nuevo elemento al par ordenado anteriormente detallado, de modo que se crea un nuevo par ordenado que obedece al formato:

(*columna, fila, puntaje*)

Y, cuando se han evaluado todos los candidatos viables, se realiza un llamado a la función selección, a la cual se le entrega la lista de los candidatos evaluados.

- d) Función selección: Está función realiza una tarea muy simple, ya que únicamente debe tomar el último elemento de cada candidato (puntaje) e ir

comparándolo con el del resto de los candidatos, con el fin de buscar el candidato con el mejor puntaje y brindar a la función solución.

- e) Función solución: Finalmente la función solución recibe el candidato mejor valorado y, tomando la columna y fila de dicho candidato, modifica la matriz intercambiando el 0 que se encuentra en esa posición por el valor con el que se encuentra jugando la computadora (1 o 2 según corresponda).

Una vez finalizado el proceso del algoritmo goloso, la computadora habrá jugado su turno y se retorna la matriz actualizada para que esta pueda ser debidamente mostrada en la interfaz gráfica de usuario y que el juego continúe normalmente.

Ejemplo de funcionamiento para una matriz 4x4:

Dada la matriz:

0	0	0	0
2	0	1	2
2	1	2	1
1	1	1	2

Donde el jugador es representado con el número 1 y la computadora se representa con el número 2.

Dicha matriz puede ser representada mediante una lista, de modo que la matriz en forma de lista resultaría tal que:

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

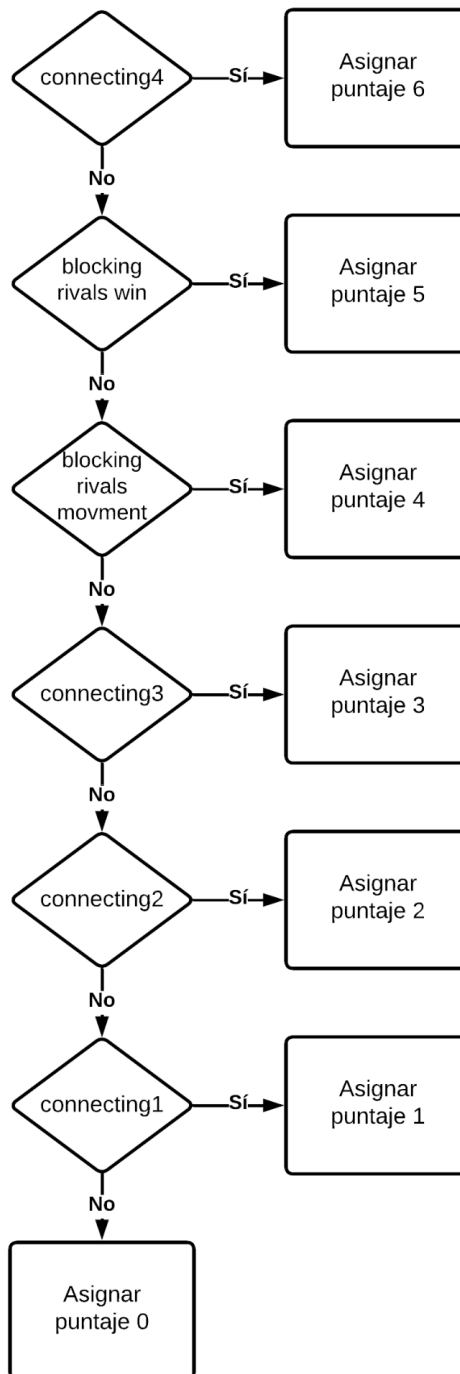
- a) Función candidatos y viabilidad:

Para este caso, se seleccionarán las casillas (1,1), (2,2), (3,1) y (4,1) como candidatos reales para posicionar una ficha. Por lo que el dato que se enviará a la función objetivo será una lista con dichos pares ordenados, tal que:

'((1 1) (2 2) (3 1) (4 1))

b) Función objetivo:

Luego, la función objetivo tomará los pares ordenados de uno por uno, evaluandolos según las pruebas detalladas anteriormente, de modo que:



Así, se asignan los siguientes puntajes:

Candidatos	Puntaje	Justificación
(1 1)	3	Conecta 3 fichas en fila.
(2 2)	4	Bloquea una posible fila de 3 fichas del jugador.
(3 1)	2	Conecta 2 fichas en fila.
(4 1)	5	Bloquea la victoria del jugador.

De este modo, se envía a la función selección la siguiente lista con elementos según el formato (columna, fila, puntaje):

'((1 1 3) (2 2 4) (3 1 2) (4 1 5))

c) Función selección:

A continuación, se seleccionará el candidato con el mayor puntaje y enviará a la función solución el par ordenado en el que se encuentra el mejor candidato, tal que:

'(4 1)

d) Función solución:

Finalmente la función solución toma dicho par ordenado y, utilizando la función set-element-in, reemplaza el 0 ubicado en la posición (4, 1) y lo reemplaza por un 2 (valor con el que juega la computadora). Así, se obtiene la matriz resultante con el cambio una vez realizado:

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

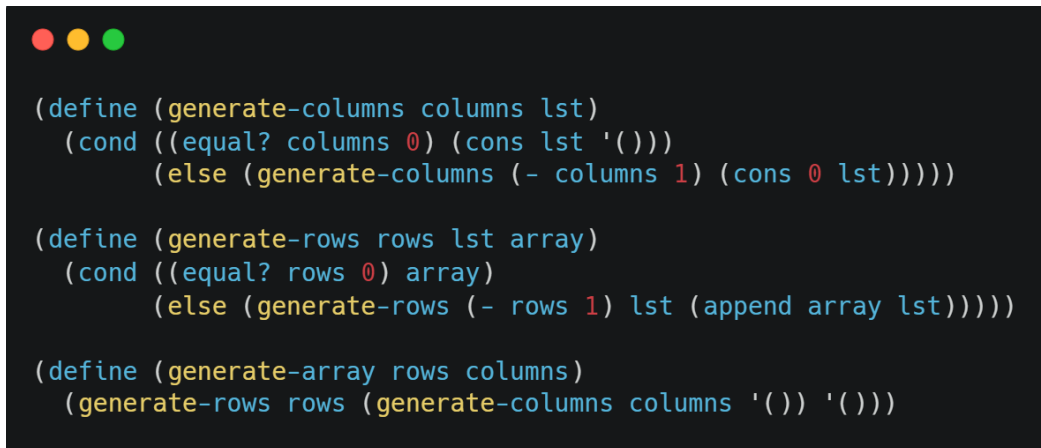
Donde esta matriz puede representarse tal que:

0	0	0	2
2	0	1	2
2	1	2	1
1	1	1	2



## 1.2 Descripción de las funciones implementadas

- a) Función `generate-array`: Esta función es vital para el funcionamiento del programa ya que es la que nos genera la matriz con la que se representa el tablero de juego. Esta función usa `generate-rows` y `generate-columns` de auxiliares. La función `generate-columns` genera una única fila de ceros, está con la cantidad de ceros según la cantidad de columnas especificadas. Posteriormente, cuando se tiene una única fila de ceros acorde a la cantidad de columnas, esta se va clonando dentro de otra lista, esto la cantidad de veces que las filas nos especifique. De esta forma, nos queda una lista con sublistas que representan la totalidad de la matriz vacía para empezar a jugar.



```
(define (generate-columns columns lst)
  (cond ((equal? columns 0) (cons lst '()))
        (else (generate-columns (- columns 1) (cons 0 lst)))))

(define (generate-rows rows lst array)
  (cond ((equal? rows 0) array)
        (else (generate-rows (- rows 1) lst (append array lst)))))

(define (generate-array rows columns)
  (generate-rows rows (generate-columns columns '()) '()))
```

- b) Función `is-full?`: Esta función también es de gran importancia en la lógica del juego ya que nos indica la situación de empate, donde el tablero se llena y no se puede seguir jugando. Esta función utiliza la función auxiliar “`is-full-row?`” la cual revisa si hay algún cero en la fila dada. Esto debido a que utilizamos el 0 para representar vacío. De esta forma, “`is-full?`” analiza la primera fila en el tablero para saber si este ya está completamente lleno. Cabe destacar que solo revisa

la primera fila ya que esta siempre va a ser la última en llenarse.

```
(define (is-full-row? row)
  (cond
    ((null? row) #t)
    ((equal? (car row) 0) #f)
    (else (is-full-row? (cdr row)))))

(define (is-full? matrix)
  (cond
    ((equal? #t (is-full-row? (car matrix))) #t)
    (else #f)))
```

- c) Función add-token: Esta función sirve para realizar una jugada dentro del tablero, por lo que manipula la matriz indicando la jugada especificada. Esta función utiliza 3 auxiliares para realizar este proceso: “without-tail”, “change-element” y “is-empty-row?”. La funcionalidad de estas se especifica en la documentación interna. Teniendo esto en cuenta, esta función primero revisa si la posición de la columna que se especifica en la última fila está vacía. En dado caso, se pone una ficha ahí. En caso contrario, se quita la última fila de la matriz con ayuda de “without-tail” y se vuelve a evaluar la “nueva última fila”. De esta forma se sigue hasta que se pueda poner una ficha.

```
(define (add-token column player matrix matrixP)
  (cond
    ((is-empty-row? column (list-ref matrix (- (length matrix) 1))) (append (without-tail matrix)
    (list(change-element column player (list-ref matrix (- (length matrix) 1)) '()) matrixP))
    (else (add-token column player (without-tail matrix) (append (list(list-ref matrix (- (length
matrix) 1))) matrixP)))))
```

- d) Función win?: Al igual que “is-full?”, esta función comprueba uno de los posibles finales de la partida, que alguien gane. Esta función utiliza varias funciones auxiliares que en su generalidad revisan la cantidad de fichas que hay de dado jugador hacia cierta dirección. Estas direcciones son: vertical, horizontal,

diagonal par y diagonal impar. De esta forma si se forman al menos 4 fichas seguidas en una de estas direcciones se gana.

```
(define (win? column row matrix player)
  (cond
    ((or (odd-diagonal-win column row matrix player) (pair-diagonal-win column row matrix player)
      (vertical-win column row matrix player) (horizontal-win column row matrix player)) #t)
    (else #f)))
```

- e) Función get-element-in: Esta función realiza un trabajo muy simple, sin embargo, para un programa como este, dicho trabajo es muy provechoso, ya que esta función se encarga de recibir una matriz y un par coordenado (una columna y una fila) y retorna el valor que se encuentra en esa posición específica de dicha matriz, de este modo, se aligera el proceso de consulta para el valor que se encuentra en cada posición de la matriz.

Para ello la función get-element-in recibe la columna, fila y matriz correspondientes, posteriormente comienza un llamado recursivo, en el cual “descarta” filas a la vez que reduce el valor del parámetro “row”. Una vez que dicho parámetro alcanza el valor 1 significa que ha encontrado la fila que buscaba, por lo que llama a la función auxiliar suministrando la fila (sublista de elementos encontrada recientemente) y el número de columna en la que se encuentra el elemento que se desea consultar.

Finalmente la función auxiliar get-element-in-aux toma la sublista y el número de columna dados y procede a “iterarse” recursivamente, descartando el primer elemento de la sublista y reduciendo en una unidad el valor de la columna brindada. En el momento en el que el valor del parámetro columna obtiene un valor de 1, significa que el elemento que se desea consultar corresponde al primer elemento de la sublista (fila dada al iniciar la función auxiliar y que ha sido editada con cada llamado recursivo), por lo cual, únicamente es necesario realizar un (car) para brindar el resultado consultado.

```
(define (get-element-in column row matrix)
  (cond ((or (null? matrix) (> row (length (car matrix))) (= row 0))
        null)
        ((> row 1) (get-element-in column (- row 1) (cdr matrix)))
        (else (get-element-in-aux column (car matrix)))))

(define (get-element-in-aux column row)
  (cond ((or (null? row) (> column (length row)) (= column 0)) null)
        ((> column 1) (get-element-in-aux (- column 1) (cdr row)))
        (else (car row))))
```

- f) Función set-element-in: Del mismo modo, esta función también realiza y una labor significativa para este programa, ya que recibe una matriz, un par coordinado y un valor, donde dicho valor será el que sustituya al anterior valor ubicado en las coordenadas dadas de la matriz proporcionada, de modo que se hace más ligera la tarea de modificar una posición concreta dentro de la matriz. En este caso, esta función funciona con una metodología similar a la de la función get-element-in, sin embargo, esta función no descarta las filas o elementos hasta llegar a la posición consultada, sino que los almacena en una matriz nueva creada a partir de la anterior.
- Cuando encuentra el elemento que busca (utilizando la metodología de la función get-element-in y su función auxiliar), procede a insertar el nuevo elemento en la matriz creada a partir de la anterior, de modo que descarta el valor anterior que se encontraba en la posición dada y lo reemplaza por el nuevo valor recibido como parámetro (newElement).
- Finalmente, termina de recorrer la matriz con el objetivo de seguir construyendo la nueva matriz; cuando la matriz original se encuentra vacía significa que ya se ha reconstruido la nueva matriz, por lo tanto se procede a finalizar el proceso de la función set-element-in, retornando la nueva matriz construida a partir de la anterior.

```
(define (set-element-in column row newElement matrix)
  (set-element-in-aux column row newElement matrix '()))

(define (set-element-in-aux column row newElement matrix newMatrix)
  (cond ((null? matrix) newMatrix)
        ((> row (length (car matrix))) null)
        ((> row 1) (set-element-in-aux column (- row 1) newElement (cdr matrix) (append newMatrix (list (car matrix)))))
        ((= row 1) (set-element-in-aux column (- row 1) newElement (cdr matrix) (append newMatrix (list (set-element-in-aux2 column (car matrix) newElement '())))))
        (else (set-element-in-aux column (- row 1) newElement (cdr matrix) (append newMatrix (list (car matrix)))))))

(define (set-element-in-aux2 column row newElement newRow)
  (cond ((> column (length row)) null)
        ((null? row) newRow)
        ((> column 1) (set-element-in-aux2 (- column 1) (cdr row) newElement (append newRow (list (car row)))))
        ((= column 1) (set-element-in-aux2 (- column 1) (cdr row) newElement (append newRow (list newElement))))
        (else (set-element-in-aux2 (- column 1) (cdr row) newElement (append newRow (list (car row))))))
```

### 1.3 Descripción de las estructuras de datos utilizadas

Matriz (lista de listas): Para el desarrollo de este programa se optó por implementar una matriz o lista de listas, con el fin de que dicha matriz representara al tablero de juego. Para hacer más dinámicas las comprobaciones y facilitar el manejo de datos de esta estructura, se estableció una simbología numérica que permitiera representar cada espacio de la matriz como un número, tal que:

Valor	Simbología
0	Representa a todo aquel espacio de la matriz donde no hay una ficha (ni del jugador ni de la computadora).
1, 2	Representa a todo aquel espacio de la matriz donde haya una ficha tanto del jugador como de la computadora (el valor depende del color de ficha, el cual es seleccionado por el jugador antes de iniciar la partida y por tanto, un valor variable).

De este modo, a la hora de iniciar la partida se cuenta con una matriz llena de ceros en todas sus posiciones, no obstante, conforme avance la partida, dichos ceros serán reemplazados por un 1 o un 2, según quien juegue en las diferentes casillas del tablero.

## 1.4 Problemas sin solución

Todos los problemas que se encontraron durante el desarrollo del proyecto fueron solucionados.

## 1.5 Problemas encontrados

- **Importación de archivos:**

Hubo problemas al enlazar la lógica del juego (*logic.rkt*) con la interfaz gráfica (*gui.rkt*), a pesar que ambos archivos estaban contenidos en la misma carpeta, se consultó en internet y el problema se resolvió al quitarle a la lógica del juego, la línea donde se indica que el archivo usará la variante normal de Racket (`#lang racket`). Se recomienda que el archivo que se desea importar esté en el mismo lugar o carpeta de aquel que lo va a utilizar y que ambos archivos sean independientes entre sí.

- **Dibujar fichas y tablero:**

En la interfaz gráfica las fichas y el tablero se dibujaron con la clase *bitmap%* del kit de herramientas de dibujo de Racket y hubieron ciertos inconvenientes sobre cómo actualizar el tablero cuando el jugador o la computadora agregaban una nueva ficha y cómo colocar ese dibujo actualizado en la ventana de juego. Consultando la documentación de racket, se encontró que la clase *message%* permite en uno de sus parámetros (*label*), pasar la ruta de la imagen que se desea cargar por medio de un mapa de bits (*read-bitmap*), entonces se decidió que las fichas y el tablero creadas con *bitmap%* se guardarán en la carpeta *resources* del proyecto y cuando fuera necesario, agregarlas por medio de *message%*.

Si lo que se busca es adjuntar imágenes estáticas, se recomienda usar este método, por lo contrario, si se necesita manipular variables de la imagen (dimensión, posicionamiento, filtros, entre otros), se sugiere buscar un método alternativo.

- **Congelación de interfaz gráfica:**

Cuando se agregaba una nueva ficha al tablero, para dar el efecto que estaba cayendo hasta el fondo de la columna se esperaba un pequeño tiempo cada vez que la ficha iba bajando entre filas, por eso en un principio se utilizó la función *sleep* que interrumpe la ejecución del juego durante un tiempo específico, sin embargo, esto hacía que la interfaz gráfica se congelara. Se investigó un poco y se encontró que la alternativa para que eso no pasara era usar *sleep/yield* que bloquea durante al menos el número especificado de segundos, mientras maneja eventos si el subproceso actual es el subproceso del controlador del espacio de eventos actual.

- **Tablero con mayor cantidad de filas que columnas:**

Cuando se jugaba con un tablero de mayor cantidad de filas que de columnas, el programa fallaba a la hora de pasar por el proceso del algoritmo goloso, sin embargo, dicho problema tuvo una solución sencilla, ya que únicamente fue necesario realizar una breve revisión para hallar que existía un fallo en la codificación de las funciones *set-element-in* y *get-element-in*, ya que se estaba leyendo incorrectamente el número de filas de la matriz dada.

## 1.6 Plan de actividades

Descripción de la tarea	Tiempo estimado	Responsable	Fecha de entrega
Leer la documentación sobre el desarrollo de interfaces gráficas en Racket	4 horas	-Emanuel Marín	Lunes 27 de febrero
Crear las principales ventanas de la interfaz	2 horas	-Emanuel Marín	Lunes 27 de febrero
Cargar ficha azul y blanca en la ventana de inicio junto con un radio button para poder seleccionar una de ellas	1 hora	-Emanuel Marín	Martes 28 de febrero
Crear ventana de diálogo para poder escoger dimensiones del tablero diferentes a 8x8 y 16x16	1 hora	-Emanuel Marín	Martes 28 de febrero

Cargar el tablero en la ventana de juego dada una matriz mxn	4 horas	-Emanuel Marín	Martes 28 de febrero
Cambiar el tamaño de las celdas del tablero	1 hora	-Emanuel Marín	Miércoles 1 de marzo
Cargar el nombre del jugador y las dimensiones del tablero para comenzar la partida	2 horas	-Emanuel Marín	Jueves 2 de marzo
Añadir fichas al tablero	2 horas	-Emanuel Marín	Jueves 2 de marzo
Integrar el algoritmo codicioso a la interfaz	2 horas	-Emanuel Marín	Sábado 4 de marzo
Agregar validación de ganador en la interfaz	3 horas	-Emanuel Marín	Sábado 4 de marzo
Investigar sobre cómo crear un ejecutable del proyecto y probarlo	1 hora	-Emanuel Marín	Sábado 4 de marzo
Validar que el jugador ya no pueda agregar fichas a una columna que ya esté llena y mostrar una ventana de ello	2 horas	-Emanuel Marín	Lunes 6 de marzo
Mostrar en la interfaz cuando hay un empate entre el jugador y la PC, es decir, cuando el tablero está lleno	1 hora	-Emanuel Marín	Lunes 6 de marzo
Agregar sonido cuando se le da click a los botones de la interfaz y cuando se agrega una nueva ficha al tablero	1 hora	-Emanuel Marín	Jueves 9 de marzo
Crear animación cuando la ficha se agrega al tablero	2 horas	-Emanuel Marín	Viernes 10 de marzo
Funciones iniciales de la lógica interna del juego	3 días	-Gabriel Chacon	Jueves 2 de marzo
Finalizar la creación de funciones que permitan al usuario disfrutar de una experiencia de juego óptima	3 días	-Gabriel Chacon	Domingo 5 de marzo
Documentación interna de la funciones realizadas	1 hora	-Gabriel Chacon	Miércoles 8 de marzo



Documentación externa y entrega de la tarea	2h	-Gabriel Chacon	Sabado 11 de marzo
Desarrollar un conjunto de funciones auxiliares para la matriz (tablero de juego), que permita consultar y modificar una posición específica de dicha matriz.	1 hora	-Andrés Rodríguez	Jueves 2 de marzo
Implementar una función candidatos a modo de etapa inicial para el algoritmo goloso.	1 hora	-Andrés Rodríguez	Viernes 3 de marzo
Codificar las funciones viabilidad y objetivo para el algoritmo codicioso.	3 horas	-Andrés Rodríguez	Sabado 4 de Marzo
Desarrollar las funciones selección y solución, de modo que el algoritmo goloso se encuentre totalmente funcional.	2 horas	-Andrés Rodríguez	Domingo 5 de marzo
Acondicionar el proceso del algoritmo codicioso para garantizar un resultado que se encuentre en un formato óptimo para su posterior lectura e interpretación por parte de la interfaz gráfica de usuario.	1 hora	-Andrés Rodríguez	Lunes 6 de marzo

## 1.7 Conclusiones

- Se reafirmó y amplió el conocimiento del paradigma de programación funcional creando un juego con el lenguaje de programación Racket.
- Las listas como estructura de datos facilitaron el manejo y flujo de la información necesaria para la ejecución del juego.
- The Racket Graphical Interface Toolkit es una biblioteca que contiene lo necesario para crear interfaces básicas amigables con el usuario.

## 1.8 Recomendaciones

- Usar el paradigma de diseño algorítmico divide y vencerás (DYV), creando funciones auxiliares siempre que sea necesario, tratando que la función definida haga una sola cosa en particular.
- Si se quiere agregar animaciones en la interfaz, se recomienda utilizar la función *sleep/yield* ya que no congela la ventana y permite seguir manejando eventos.
- Crear ejecutables con DrRacket, ya que es sencillo de realizar y ofrece varias formas para hacerlo, en lugar de investigar cómo se haría en la ventana de comandos.

## 1.9 Bibliografía

*Four In A Row Online - Connect all 4* | Coolmath Games. (s. f.).

<https://www.coolmathgames.com/0-four-in-a-row>

Including an external file in racket. (2011, 26 enero). Stack Overflow.

<https://stackoverflow.com/questions/4809433/including-an-external-file-in-racket/4937138>

*The Racket Graphical Interface Toolkit*. (s. f.). <https://docs.racket-lang.org/gui/>

*The Racket Drawing Toolkit*. (s. f.). <https://docs.racket-lang.org/draw/index.html>

Wikipedia contributors. (2022, 23 noviembre). *Greedy algorithm*. Wikipedia.

[https://en.wikipedia.org/wiki/Greedy\\_algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm)

## 1.10 Bitácora

Fecha	Presentes	Duración	Avance
27/02/2023	-Gabriel Chacon -Emanuel Marín -Andrés Rodriguez	1 hora	Nos reunimos para leer con atención la especificación y empezamos a establecer roles. Se empezó la documentación y se generó el repositorio.
03/03/2023	-Gabriel Chacón -Andres Rodriguez	2 horas	Se hizo una reunión para aclarar conceptos del algoritmo greedy y coordinar próximas funciones a realizar.

04/03/2023	-Gabriel Chacon -Emanuel Marín -Andrés Rodríguez	1 hora	Nos reunimos para revisar la funciones que creamos, principalmente las del algoritmo goloso y discutimos un poco sobre cómo integrarlas con la interfaz gráfica, además vimos que otras funciones sería necesario crear.
27/02/2023	-Emanuel Marín	4 horas	Realicé la lectura de la documentación oficial de Racket para el desarrollo de interfaces gráficas, hice énfasis en las secciones necesarias para la implementación de la tarea (ventanas, widgets, manejo de evento y de archivos, entre otros).
27/02/2023	-Emanuel Marín	1 hora	El día de hoy, creé el esqueleto de una interfaz funcional a manera de ejemplo para corroborar lo aprendido hasta el momento (creación y distribución de contenedores, manejo de eventos, excepciones, creación e instanciación de clases, entre otros).
27/02/2023	-Emanuel Marín	2 horas	En el transcurso de la tarde trabajé en la ventana de inicio y en el acceso de esta a la ventana de información y del juego.
28/02/2023	-Emanuel Marín	6 horas	El día de hoy cargué en la ventana de inicio la ficha de color azul y blanca que el jugador puede seleccionar antes de iniciar la partida. También trabajé en poder cargar de forma dinámica una tablero de dimensión mxn a partir de una matriz dada.
01/03/2023	-Emanuel Marín	1 hora	Hoy solo realicé un pequeño cambio al tamaño de las celdas que conforman el tablero para que el de 16x16 no sea tan grande en la ventana de juego.
02/03/2023	-Emanuel Marín	4 horas	El día de hoy cargué el nombre del jugador en la ventana de juego, además de eso también pude agregar fichas en el tablero y ver como este se actualiza.
03/03/2023	-Emanuel Marín	2 horas	Hoy sólo hice algunos cambios para mejorar la interfaz, cambios en el tamaño de las ventanas, la distribución de los widgets en ellas y sus bitmaps.
04/03/2023	-Emanuel Marín	6 horas	Por la noche me enfoqué en integrar con la interfaz el algoritmo codicioso y la validación de quién ganó entre el jugador y la PC, mostrando

			una ventana de diálogo para ello. Por último me puse a investigar sobre cómo crear ejecutable con Racket y probé hacer uno del proyecto desde el editor Dr Racket.
09/03/2023	-Emanuel Marín	1 hora	El día de hoy decidí agregar sonido al dar click en cualquiera de los botones de la interfaz y al agregar una nueva ficha al tablero. También quería regular el volumen o silenciarlo del todo pero decidí dejarlo para mañana quizás.
10/03/2023	-Emanuel Marín	2 horas	Por la tarde me enfoqué en crear el efecto de dejar caer la ficha sobre una columna del tablero y al final decidí quitarle el sonido a la interfaz.
27/02/2023	-Gabriel Chacón	1 hora	Se empezaron a realizar las funciones básicas de la lógica general del juego. Se logró terminar la función que genera la matriz de las dimensiones especificadas con sus auxiliares.
28/02/2023	-Gabriel Chacón	2 horas	Se terminaron las funciones de jugar ficha y la función que comprueba que el tablero esté lleno de fichas, o sea, empate.
02/03/2023	-Gabriel Chacon	2 horas	Se realizaron funciones complementarias y auxiliares para el correcto funcionamiento de la lógica y para ayudar a enlazar estas funciones con la interfaz.
08/03/2023	-Gabriel Chacon	1h	Se realizó la documentación interna de las diferentes funciones implementadas para la lógica del juego.
11/03/2023	-Gabriel Chacon	1h	Se terminaron de revisar detalles finales y documentación para la entrega final del trabajo
02/03/2023	-Andrés Rodríguez	1 hora	Se codificaron dos funciones básicas que permiten consultar y modificar una posición precisa de la matriz.
03/03/2023	-Andrés Rodríguez	1 hora	Se implementó la función candidatos para el algoritmo goloso.
04/03/2023	-Andrés Rodríguez	2 horas	Se desarrollaron las funciones viabilidad y objetivo como parte del proceso del algoritmo codicioso.
05/03/2023	-Andrés Rodríguez	2 horas	Se finalizó la creación de funciones que forman parte del algoritmo goloso, desarrollando las

			funciones selección y solución.
06/03/2023	-Andrés Rodríguez	1 hora	Se adaptó el resultado del algoritmo codicioso con el objetivo de brindar a la interfaz gráfica de usuario una solución con un formato apropiado para su posterior lectura e interpretación.
06/03/2023	-Andrés Rodríguez	1 hora	Se realizó una sesión de pruebas al programa actual, en la cual el programa mostró un óptimo funcionamiento.
07/03/2023	-Andrés Rodríguez	2 horas	Se complementó la documentación externa, específicamente los apartados de algoritmos desarrollados, funciones implementadas y estructuras de datos desarrolladas.
08/03/2023	-Andrés Rodríguez	1 hora	Se solucionó un bug relacionado con las dimensiones del tablero de juego, el cual provocaba que el programa fallara cuando se jugaba con un tablero con una mayor cantidad de filas que columnas.