



Instituto Tecnológico de Costa Rica.

Área de Ingeniería en Computadores.

Lenguajes Intérpretes y Compiladores.

Profesor: Marco Rivera.

Grupo 02.

Tarea 1: 4Line.

Estudiantes:

- Santiago Brenes Torres (2019063875)
- Tomás Segura Monge (2018099729)



Documentación.

A continuación se presenta la documentación externa del proyecto 4Line que se puede encontrar en [GitHub](#). El esquema del documento es el siguiente.

- 1 - Documentación del código.
 - 1.1 - Descripción del algoritmo de solución.
 - 1.2 - Descripción de las funciones
 - 1.2.1 - Funciones para la lógica.
 - 1.2.2 - Funciones para la interfaz.
 - 1.3 - Estructuras de datos usadas.
 - 1.3.1 - Matriz de la interfaz gráfica.
 - 1.3.2 - Matriz de la lógica.
 - 1.4 - Problemas sin solución.
 - 1.4.1 - Columna llena.
 - 1.4.2 - Click en botones deshabilitados.
 - 1.5 - Problemas encontrados.
 - 1.5.1 - Matrices transpuestas.
 - 1.5.2 - Actualización del tablero.
 - 1.5.3 - Verificar ganador.

- 1.6 - Conclusiones.
- 1.7 - Recomendaciones.
- 2 - Bitácoras.
 - 2.1 - Bitácora de reuniones.
 - 2.2 - Bitácora de Santiago.
 - 2.3 - Bitácora de Tomás.

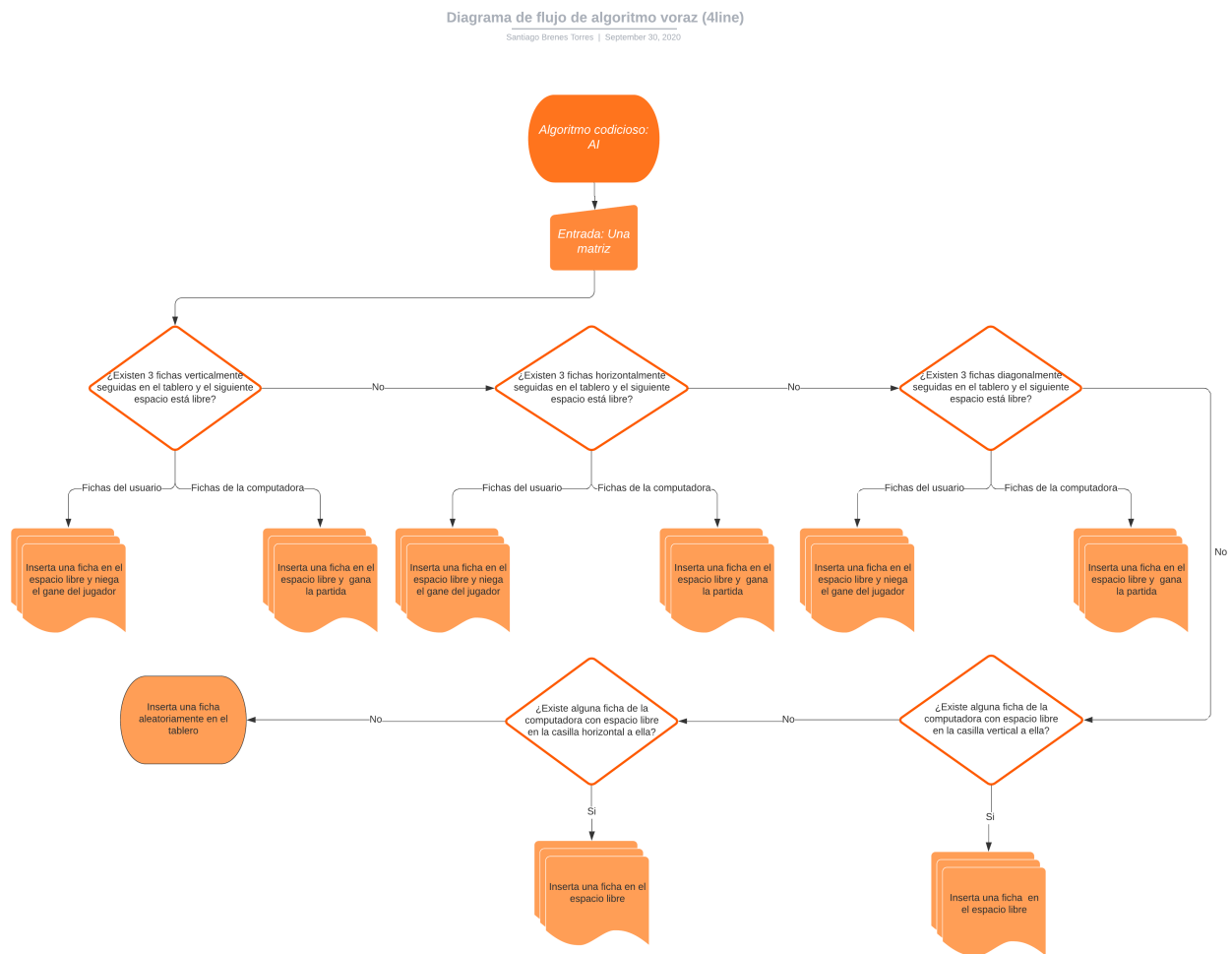
1.1 - Descripción del algoritmo de solución.

El algoritmo voraz se divide en 5 funciones. Las cuales consisten en:

1. Las primeras 3 revisan si existen 3 fichas seguidas en línea: vertical, horizontal o diagonal, en el caso de que estas fichas sean encontradas, las funciones verificarán si la ficha siguiente a estas 3 (la que completaría 4 fichas en línea) se encuentra vacía, en el caso que sea cierto, se insertará una ficha que:
 - Si las 3 fichas son del usuario, el algoritmo evitará que este gane.
 - Si las fichas son de la computadora, el algoritmo insertará la ficha para que la computadora gane.
2. Si no se encuentra algún candidato cercano a ganar, el algoritmo llamará a otras dos funciones. La primera recorre la matriz en busca de una ficha de la computadora:
 - Si la encuentra, se asegura de que haya un espacio vacío encima de esta, si este caso es verdadero, el algoritmo insertará una ficha en la columna donde se encontró la ficha.
 - Si una ficha del usuario se encuentra por encima de la ficha encontrada, o la columna se encuentra llena, el algoritmo continuará buscando fichas de la computadora en las columnas siguientes de la matriz.
 - Si no existen fichas del computador con espacios vacíos por encima de estas, el algoritmo llamará a la segunda función.
3. Esta última función, al igual que la anterior, recorre la matriz en busca de fichas del computador, en el momento que encuentra una, la función revisa que el espacio, en la misma fila, de la columna siguiente se encuentre vacío.
 - Si este último enunciado es verdadero, verifica que exista alguna ficha por debajo del espacio vacío (asegurando que la la ficha insertada caiga horizontal a la ficha seleccionada), y que no hayan fichas por encima de este.

- Si se cumplen estas condiciones, el algoritmo insertará una ficha en la columna donde se encuentra el espacio vacío, horizontal a la ficha encontrada.
- Si las condiciones no se cumplen, continuará buscando una ficha del computador que las cumpla.

4. En el caso de no se cumpla ninguna de las condiciones anteriores, el algoritmo insertará una ficha en una columna elegida al azar.



1.2 - Descripción de las funciones implementadas.

1.2.1 - Funciones para la lógica.

(4line rows columns)

Descripción: Crea una matriz a partir de estas.

1. Revisa que el número de filas y columnas se encuentren dentro del rango definido.
2. Llama a la función `createBoardMatrix`, la cual agrega la cantidad de columnas a la matriz, mientras que llama a otra función que va agregando las filas como ceros.

Parámetros:

- `rows` : Número de filas deseado.
- `columns` : Número de columnas deseado.

Output: Retorna la matriz lógica de la forma:

```
'((0 0 0 0)
  (0 0 0 0)
  (0 0 0 0)
  (0 0 0 0))
```

(len lista)

Descripción: Analiza el largo de una lista.

Parámetros:

- `lista` : Una lista.

Output: Largo de la lista.

(mainAux player0n)

Descripción: Se encarga de manejar los turnos de los jugadores. Administra la lógica. Es la función que llama a `AI`.

Parámetros:

- `player0n` : Jugador (1) o PC (2).

Output: Nada, únicamente administra la lógica.

(insertCoin column player matrix)

Descripción: Inserta una ficha, puede ser del cliente (1) o de la computadora (2), en el último espacio disponible de una columna en la matriz. Hace uso de la función auxiliar `insertCoin_aux`.

1. Identifica la columna en la que se debe insertar la ficha, mediante un contador, `cdr` y recursión.
2. Al llegar a la columna, la envía a una función auxiliar, la cual verifica cual es el último elemento que no contiene una ficha.
3. Agrega la ficha en el último espacio vacío.
4. Reconstruye la fila enviada mediante `cons`.
5. Vuelve a la función principal con la ficha agregada en la lista.
6. Reconstruye la matriz mediante `cons`.

Parámetros:

- `column` : Columna en la que se desea insertar la ficha.
- `player` : Jugador (1) o PC (2).
- `matrix` : Matriz en análisis.

Output: Retorna la matriz con la ficha insertada.

(insertCoinRow column player matrix)

Descripción: Retorna la fila en la que cae la ficha insertada en la matriz. Hace uso de la función auxiliar `insertCoinRow_aux`.

1. Identifica la columna en la que se debe insertar la ficha, mediante un contador, `cdr` y recursión.
2. Al llegar a la columna, la envía a una función auxiliar con un contador, la cual verifica cual es el último elemento que no contiene una ficha, mientras va sumando 1 al contador por cada vez que hace la llamada recursiva.
3. Al llegar al último elemento vacío en la lista, la función devuelve al contador que indica en qué número de fila se encuentra la ficha agregada.

Parámetros:

- `column` : Columna en la que se desea insertar.
- `player` : Jugador (1) o PC (2).
- `matrix` : Matriz en análisis.

Output: Fila en la que se debe insertar la ficha.

(checkHorizontales matrix)

Descripción: Verifica si la computadora o el cliente se encuentra cerca de ganar (4 fichas en línea) horizontalmente. Hace uso de la función `checkHorizontalesAux`.

1. La función principal llama a una auxiliar, pasando como parámetros la matriz, el índice de columna y fila en la que inicia el recorrido de la matriz (1 y 1), y dos contadores para las repeticiones de las fichas
2. La función auxiliar recorre la matriz mediante los índices de filas y columnas, cuando encuentra una ficha agrega un punto al contador del jugador o de la computadora.
3. Revisa si la siguiente ficha que se encuentra horizontalmente es igual a la anterior, si es verdadero, agrega otro punto al contador y vuelve a revisar si la siguiente es igual, si la condición resulta falsa, continúa el recorrido de la matriz y reinicia los contadores de puntos.
4. Cuando encuentra 3 fichas iguales, seguidas y horizontalmente, retorna el número de columna siguiente a la última de las 3 fichas y el número de jugador a la que pertenecen las 3 fichas, ambas en una lista.
5. Si no las encuentra retorna 0.

Parámetros:

- `matrix` : matriz en análisis.

Output: Retorna el jugador que está cerca a ganar. Jugador (1), PC (2) o 0 .

(checkHorizontal matrix)

Descripción: Encuentra una ficha de la computadora (2) y verifica si se encuentra un espacio vacío horizontalmente. Hace uso de la función auxiliar `checkHorizontalAux`.

1. La función principal llama a una auxiliar, pasando como parámetros la matriz y el índice de columna y fila en la que inicia el recorrido de la matriz (1 y 1).
2. La función auxiliar recorre la matriz mediante los índices de filas y columnas.
3. Si encuentra una ficha revisa si el espacio siguiente, horizontalmente, encuentra libre, si esto es verdadero, verifica que exista una ficha por debajo del espacio libre, si esto

también es verdadero, la función retorna el índice de la columna en la que se encuentra el espacio libre.

4. Si recorre toda la matriz y no se cumplen las condiciones anteriores, retorna 0.

Parámetros:

- `matrix` : matriz en análisis.

Output: Número de columna disponible para colocar una ficha.

(checkVerticales matrix)

Descripción: Verifica si la computadora o el cliente se encuentra cerca de ganar (4 fichas en línea) verticalmente. Recorre las listas de la matriz ya que estas representan las columnas. Hace uso de las funciones auxiliares `checkVerticalesAux` y `checkVerticalesAux2`.

Parámetros:

- `matrix` : matriz en análisis.

****Output:**** Número de fila disponible para colocar una ficha.

(checkVertical matrix)

Descripción: Busca fichas solas de la PC y analiza si está disponible el espacio encima para colocar una ficha. Hace uso de la función auxiliar `checkVerticalAux`.

Parámetros:

- `matrix` : Matriz en análisis.

Output: Número de columna en la que se puede colocar una ficha.

(checkDiagonales matrix)

Descripción: Función auxiliar para el algoritmo AI que permite analizar las diagonales de una matriz con el fin de saber si un jugador está cercano a ganar. Hace uso de las funciones

auxiliares `selectDiagonalesToCheck` y `checkDiagonalesAux` que le permiten analizar todas las diagonales posibles para ganar de arriba hacia abajo.

Parámetros:

- `matrix` : Matriz que desea analizar.

Output: Una lista de la forma '(jugador fila columna)' que indica el jugador que ganará en la fila y columna definidas.

(checkWinner matrix)

Descripción: Analiza una matriz para saber si algún jugador ha ganado. Analiza todas las verticales, todas las horizontales y las diagonales en el tablero. Para esto hace uso de las funciones auxiliares `checkRows`, `checkColumns` y `checkDiagonals`.

Parámetros:

- `matrix` : Matriz que desea analizar.

Output: Jugador que gana (0 , 1 , 2).

(fullColumn? lista)

Descripción: Analiza si una columna está llena o no.

Parámetros:

- `lista` : Columna.

Output: #t o #f.

1.2.2 - Funciones para la interfaz.

(createBoardButtons columns rows)

Descripción: Se encarga de crear todos los botones correspondientes al tablero. Hace uso de las funciones auxiliares `createBoardButtonsRows` y `createBoardButtonsColumns` que

construyen la matriz en filas y columnas.

Parámetros:

- `columns` : Número de columnas que escogió el usuario.
- `rows` : Número de filas que seleccionó el usuario.

Output: Despliega en la interfaz todos los botones en forma de “tablero” para que el usuario haga uso de él.

(enableButtons enable?)

Descripción: Habilita o deshabilita para el usuario todos los botones del tablero. Hace uso de las funciones auxiliares `enableButtonsRows` y `enableButtonsColumns` que recorren la matriz en filas y columnas.

Parámetros:

- `enable?` : booleano que indica si se desea habilitar `#t` o deshabilitar `#f`.

Output: Bloquea los botones en la interfaz de modo que el usuario no puede seleccionarlos.

(createHorizontalStandardPane)

Descripción: Crea un `horizontal-pane%` de la interfaz de racket servirá para el tablero ya que en él se colocan las filas de la interfaz.

Parámetros: ninguno.

Output: Despliega un pane para ubicar botones en la interfaz.

(createHoleButton parent rows column list totalRows totalColumns enabled?)

Descripción: Crea cada uno de los botones `button%` en la interfaz con los atributos que se le pasa por parámetros. Hace uso de la función auxiliar `setChoseHole` que es la función que se le asigna en el atributo `callback`.

Parámetros:

- `parent` : `container%` padre que se le asigna al botón.
- `rows` : fila a la que corresponde el botón.
- `column` : columna a la que corresponde el botón.
- `list` : booleano que indica si debe estar habilitado o no.
- `totalRows` : total de filas en la matriz.
- `totalColumns` : total de columnas en la matriz.

Output: Un botón en el tablero.

(setChoseHole row column totalRows totalColumns playerOn oldMatrix)

Descripción: Función que se le asigna a cada uno de los botones cuando se les da click. También se utiliza en el main cuando la PC inserta su ficha.

Parámetros:

- `row` : fila a la que corresponde el botón.
- `column` : columna a la que corresponde el botón.
- `totalRows` : total de filas en la matriz.
- `totalColumns` : total de columnas en la matriz.
- `playerOn` : turno de jugador 1 o 2 .
- `oldMatrix` : matriz previa a la inserción de la ficha.

Output: Cambio de estado del botón en la interfaz y en la matriz lógica.

(matrixGet row column cont list)

Descripción: nos permite obtener un botón específico dentro de la matriz gráfica. Hace uso de la función auxiliar `matrixGetAux` que juntas navegan las filas y columnas de la matriz.

Parámetros:

- `row` : fila a la que corresponde el botón.
- `column` : columna a la que corresponde el botón.
- `cont` : iniciar en 1 .

- `list : matriz.`

Output: El botón deseado.

(setTheDifferentButton matrixA matrixB playerOn)

Descripción: Recibe dos matrices transpuestas gemelas pero que varían en una de sus posiciones. Convierte el botón diferente al estado requerido. Hace uso de la función auxiliar `findDifference`.

Parámetros:

- `matrizA` : matriz transpuesta de `matrizB`.
- `matrizB` : matriz transpuesta de `matrizA`.
- `playerOn` : 1 o 2 según corresponda.

Output: Cambia de estado al botón seleccionado por el jugador.

(findDifference matrixA matrixB cont result)

Descripción: Recibe dos matrices y analiza la columna en la que exista una ficha diferente. Lo valioso de esta función es que recibe matrices transpuestas y navega las columnas y filas al revés de cómo lo haría en la otra. Hace uso de la función auxiliar `findDifferenceAux` que le permite analizar la segunda matriz.

Parámetros:

- `matrizA` : matriz transpuesta de `matrizB`.
- `matrizB` : matriz transpuesta de `matrizA`.
- `cont` : iniciar en 1.
- `result` : iniciar en 0.

Output: Columna en la que exista una diferencia.




1.3 - Estructuras de datos usadas.

1.3.1 - Matriz de la interfaz gráfica.

La IU usa una matriz para representar todos los botones del tablero de la forma:

```
'((button button button button button)
 (button button button button button)
 (button button button button button)
 (button button button button button))
```

Esta matriz permite la actualización de los botones en sus diferentes estados:

Estado	UI
Neutro (0)	
PC (1)	
Jugador (2)	

Además, a los botones se les debía asociar la matriz lógica por lo que se reescribe la lista anterior para definir la siguiente lista de ambas matrices:

```
'((button button button button button)
 (button button button button button)
 (button button button button button)
 (button button button button button) ((0 0 0 1)
                                         (0 0 2 1)
                                         (0 2 2 1)
                                         (0 0 0 1)
                                         (0 0 0 0)) )
```

La primer matriz se utiliza exclusivamente para cambiar la estética de los botones y la segunda se usa para representar los estados y comunicarse con toda la lógica y algoritmos del juego.

1.3.2 - Matriz de la lógica.

La lógica del juego utiliza una matriz que representa al tablero y sus fichas. Al iniciar el juego, se muestra de la siguiente manera:

```
'((0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0)
  (0 0 0 0 0 0 0 0))
```

Esta matriz funciona tomando como columnas del tablero las listas dentro de la matriz, por lo que al aplicar el car de la matriz, se obtiene la primera columna de esta. Mientras el largo de las listas representa la cantidad de filas del tablero.

Esta implementación de la matriz como tablero se debe a que encontramos mayor facilidad de manejar la matriz lógica en cuanto a recorridos, obtener elementos, comparar elementos y manejar las listas

Por lo que, la base del tablero se encuentra en el último elemento de todas las listas dentro de la matriz. Como se puede notar en la siguiente matriz:

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

Los elementos que se encuentran dentro de la matriz anterior son la representación de las fichas de los jugadores, las fichas de la computadora (2), las del cliente (1) y los espacios vacíos (0).

Esta implementación de la matriz como tablero se debe a que encontramos mayor facilidad de manejar la matriz lógica en cuanto a recorridos, obtener elementos, comparar elementos y manejar las listas

1.4 - Problemas sin solución.

1.4.1 - Columna llena.

Tras rellenar una columna con fichas no desarrollamos un bloqueo para que el jugador no pueda agregar más fichas en ella. Sin embargo, no se refleja en la lógica ya que este sí es un caso contemplado. El error se encuentra en no limitar al cliente.

1.4.2 - Click en botones deshabilitados.

Aunque los botones estén deshabilitados en el turno de la PC, el jugador puede dar click en uno de ellos. Esta acción se acumulará en la cola de jugadas del jugador. Creemos que este es un error de la interfaz de Racket ya que los botones están deshabilitados y este es el único acceso a ellos mismos por lo que esto no debería suceder.

1.5 - Problemas encontrados.

1.5.1 - Matrices transpuestas.

Comenzamos a construir dos matrices: una para la interfaz y otra para la lógica. Cuando nos organizamos no pensamos en que la manera de diseñar las matrices podría afectarnos pero sí lo hizo.

La matriz gráfica (la que muestra los botones del tablero) es de forma vertical:

```
'((0 0 0 0 0)
  (0 0 0 0 0)
  (0 0 0 0 0)
  (0 1 1 1 1))
```

Donde están los 1's representa la base de un tablero como lo sería en la vida real pero la matriz lógica fue diseñada de forma transpuesta:

```
'((0 0 0 1)
  (0 0 0 1)
  (0 0 0 1)
  (0 0 0 1)
  (0 0 0 0))
```

Donde los 1's representan la base de un tablero como lo sería en la vida real.

Esto representó un gran problema ya que ambas matrices deben comunicarse entre sí, además de que la matriz lógica debe estar rebotando constantemente dentro los algoritmos.

Nos dimos cuenta hasta que queríamos conectar la interfaz con la lógica. Probamos varias soluciones que no sirvieron por motivos de eficiencia o eficacia.

La solución final fue la siguiente:

findDifference

Escribimos una función `findDifference` que recibe dos matrices transpuestas y retorna la columna donde encuentra un valor diferente entre ellas. Esta función debe llamarse cada turno con el fin de que sea efectiva.

Con el valor de la columna obtenida entonces se llama a la función entonces se puede continuar el proceso de actualizar la matriz gráfica. Esta es una función que pretende ser la intérprete entre ambas matrices.

Pros	Contras
No fue necesario reconstruir alguna de las matrices.	Incrementa los procesos que se realizan en cada turno.

Recomendaciones: organizar mejor el proyecto antes de comenzar.

1.5.2 - Actualización del tablero.

Enfrentamos un serio problema para actualizar los botones del tablero.

Al diseñar el tablero, creímos que la mejor idea era usar botones para las posiciones de las fichas pero conforme avanzamos nos dimos cuenta que jugaba en nuestra contra.

Cada botón tiene un atributo `callback` que indica la función que debe cumplir el botón al dar click sobre él. En nuestro juego, cuando se da click el botón debe mostrar un cambio estético de modo que el usuario perciba el cambio. Por lo tanto debíamos pasarle la matriz. Todo bien hasta acá, la pasamos por parámetro.

Nota aclaratoria antes de continuar: los objetos de la interfaz de Racket tienen métodos `set` que permiten cambiar los atributos de ellos. Por ejemplo:

```
(send myButton set-label "Hola amigos")
```

De este modo no había problema con actualizar estos atributos cuando fuera necesario pero **no encontramos un set-callback** y aquí radica el problema.

Pero no existe un método `set` para `callback` por lo que no era posible pasarle al botón la nueva matriz lógica cuando se daba click en otro botón.

Esto quiere decir que los botones no tenían modo de comunicarse entre sí, tras intentar varios intentos de funciones propias para solucionar este problema concluimos que, si no queríamos cambiar toda la interfaz debíamos usar una variable.

Conversamos con el profesor y le pedimos permiso para definir esta variable ya que estaría principalmente asociada a la interfaz. El profesor accedió y le agradecemos muchísimo por eso ya que es fundamental para el buen funcionamiento del juego de la manera que lo diseñamos.

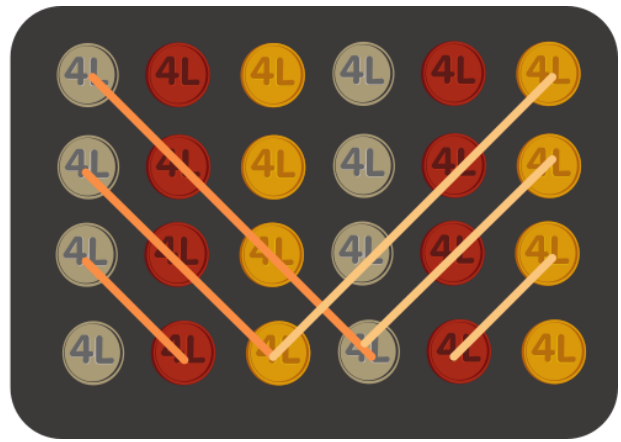
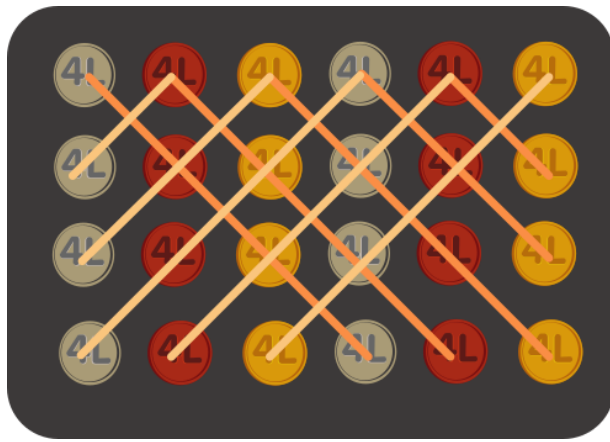
Pros	Contras
No necesitamos rediseñar toda la interfaz.	Fue necesario definir una variable y usar el método <code>set!</code> .

Recomendaciones: Investigar más opciones para interfaces de modo que se acople mejor a las condiciones del proyecto.

1.5.3 - Verificar ganador.

`checkWinner` es una función que consumo muchos recursos computacionales. La situación más complicada era revisar que alguien ganara en diagonales ya que requería de muchas vueltas más por la matriz.

La solución fue que `checkDiagonals` solamente recorriera las siguientes diagonales:



Recomendaciones: Considerar que no se debe analizar toda la matriz si se corroborará el ganador en cada turno.

1.6 - Conclusiones.

El paradigma de la programación funcional ha sido de gran utilidad a la hora de comprender e implementar, principalmente, funciones que nos plantean un panorama distinto al acostumbrado de la programación orientada a objetos, el cual ayuda a aprender sobre optimización y eficiencia mediante la recursión, utilizando únicamente funciones primitivas del lenguaje de programación utilizado, Racket.

Además, mediante la interfaz amigable de Racket, y siendo un lenguaje de programación de alto nivel, ha servido como un medio sencillo para entender la programación funcional y sus fundamentos, lo que facilitó la realización de este proyecto.

1.7 - Recomendaciones.

Primero. Debemos organizarnos mejor a la hora de hacer el planeamiento del proyecto. Por ejemplo, hacer diagramas y conversar más cómo vemos la solución cada uno de nosotros. Esto con el fin de evitar errores como programar una matriz de forma transpuesta a la del compañero lo que representó los mayores problemas durante el proyecto.

Segundo. Investigar más tecnologías que se puedan utilizar. Por ejemplo, la interfaz fue una decisión muy rápida. Esta pudo ser más meditada, ver qué otras interfaces se podían implementar en Racket.

Tercero. Poner más atención a la eficiencia de los algoritmos desarrollados. Un buen ejemplo sería la función `checkWinner` que usa fuerza bruta para revisar todas las verticales, horizontales y diagonales posibles en cada turno. Esto implicó que el juego ocupe de muchos recursos computacionales cada vez que se ejecuta el algoritmo de la PC y puede ser fácilmente optimizado solo con revisar las posiciones alrededor de la ficha que se acaba de colocar, por dar una posible solución.

2.1 - Bitácora de reuniones.

Primera reunión: 21 setiembre.

- Nos organizamos y quedamos en que Tomás haría pruebas con la interfaz de Racket y Santiago con la lógica.

Segunda reunión: 24 de setiembre.

- Tomás presentó una interfaz de un menú de inicio y una ventana para el juego.
- Santiago presentó el esqueleto para la navegación en matrices.

Tercera reunión: 28 de setiembre.

- Comenzamos a hacer pruebas para integrar una matriz lógica con una matriz gráfica de botones.
- Creemos que nos va a costar bastante integrarlas.
- Nos dimos cuenta que programamos las matrices de forma transpuesta.
- La lógica de Santiago ya inserta fichas, retorna la matriz actualizada en todas las funciones. Se puede decir que es el esqueleto del algoritmo voraz.
- Tomás escribió la función ganador que verifica si algún jugador gana.

Cuarta reunión: 29 de setiembre.

- Integramos el programa.
- Nos ha costado muchísimo debido a que las matrices están transpuestas.
- Buscamos muchos tipos de soluciones. Estamos muy cansados. Aún falta la documentación.
- Al final logramos integrar lógica e interfaz.

Reunión final: 30 de setiembre.

- Documentamos el código.
- Completamos la documentación externa.
- Pulimos detalles finales de la interfaz y el código.
- Hacemos muchas pruebas con el fin de conocer las pulgas que pueda tener.
- Sólo encontramos dos pulgas finales sin resolver. Primero, olvidamos implementar el empate. Segundo, Cuando se llena una columna de ficha, el jugador puede continuar insertando fichas ahí ya que la lógica ni la interfaz lo detiene.

2.2 - Bitácora de Santiago.

21/09/2020

- Investigar y entender los algoritmos voraces.
- Programar ejemplos de algoritmos voraces en Racket.

24/09/2020

- Repaso de funciones primitivas en Racket.
- Crear función lógica para inicializar una matriz según las filas y columnas deseadas por el usuario, dentro de los límites asignados.

26/09/2020

- Crear función lógica para “insertar una ficha” en la columna deseada.

28/09/2020

- Comienzo del algoritmo voraz
- Crear función checkDiagonal y checkVertical

29/09/2020

- Crear función principal del algoritmo voraz llamada AI
- Implementar las 5 funciones que conforman la función AI: checkDiagonales, checkVerticales, checkDiagonales, checkVertical, checkDiagonal y como última opción, insertar una ficha en una columna elegida al azar.

30/09/2020

- Arreglos finales a las funciones lógicas
- Finalización de la documentación junto a Tomás

2.3 - Bitácora de Tomás.

Jueves 24 de setiembre.

- Construcción de la interfaz básica.
- Son dos ventanas, una para definir el tablero y otra para el tablero.
- Se generan todos los botones del tablero.

Sábado 26 de setiembre.

- Construí una matriz con los botones que funciona independiente de la matriz lógica pero que se hablan entre sí. Era necesario construir esta matriz ya que los botones se generaban de bombazo y no se guardaba en memoria una forma de acceder a ellos. Por este motivo es que no podía acceder a un botón específico para cambiar su estado cuando diera click sobre él. Con la matriz de botones puedo cambiar lo que desee de cada uno de esos botones.
- Los botones se deshabilitan al momento de dar click sobre ellos.

Domingo 27 de setiembre.

- Cambié el menú inicial y la lógica de la interfaz para que fuera personalizable tanto las columnas como las filas del tablero.
- Construcción de una función que permite verificar si un jugador gana por 4 fichas en línea (columna, fila o diagonal).

28 y 29 de setiembre.

- Arreglo para que la matriz lógica pudiera comunicarse sin problemas con la matriz de la interfaz. Llevó mucho trabajo y estrés porque el tiempo apretaba mucho. En la wiki se puede leer más sobre la solución.

30 de setiembre.

- Trabajamos en la documentación y toques finales tanto en la interfaz como la lógica.

