Tarea 1

TIC TAC TOE

Paradigma de programación funcional

Instituto Tecnológico de Costa Rica

Área Ingeniería en Computadores

Lenguajes, Compiladores e intérpretes (CE3104)

Primer Semestre 2020

Preparado por:

Kevin Acevedo Rodríguez 2018148661

Hazel Martínez Loría 2018002084

Profesor:

Ing. Marco Rivera Meneses

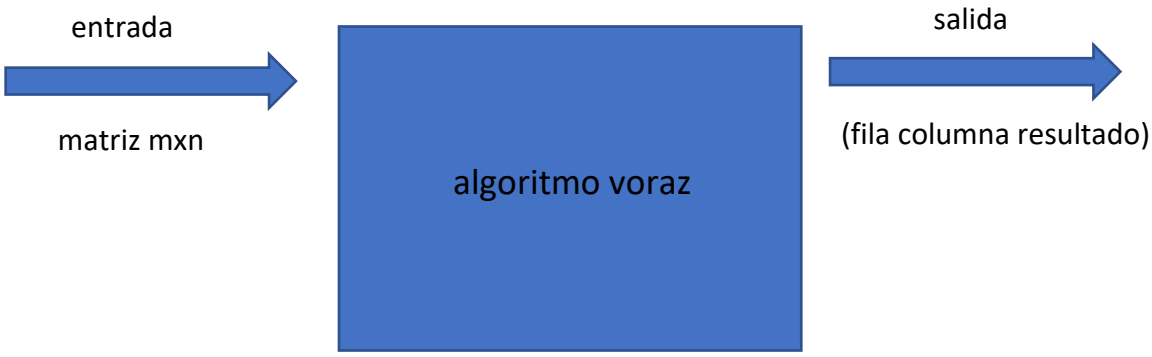
Índice

Contenido

Descripción detallada del algoritmo voraz

En esta tarea se implementó un algoritmo voraz que es capaz de **elegir la mejor posición para cada turno**, para lograrlo solo necesita recibir la matriz del turno.

Primero se va a mostrar el proceso que hay antes de que se ejecute el algoritmo y lo que debe ocurrir una vez finalizado.



En el diagrama anterior se muestra lo que recibe el algoritmo voraz y la salida que debería tener cada vez que se ejecute. Ahora se explicará la entrada y la salida del algoritmo:

Entrada: La entrada va a ser siempre una matriz mxn, esta matriz va a variar en cada turno, ya que cada vez que el jugador coloque una **X** en el tablero va a actualizar la matriz.

Las matrices que recibe el algoritmo se representan como una lista de listas:

((1 0 0) (0 0 1) (2 0 0)) → matriz 3 x 3

Ahora, los tipos de elementos posibles que puede contener la matriz son:

- 0 → un espacio vacío
- 1 → una X
- 2 → un O

Las matrices que recibe el algoritmo van desde un tamaño de 3 x 3 hasta 10 x 10.

Salida: La salida siempre será un elemento de tipo → (fila columna resultado).

En donde las dos primeras posiciones de la lista se representan a la fila y la columna en la que se encuentra el **mejor candidato** seleccionado por el algoritmo.

El **resultado** representa el estado que va a tener el juego una vez que se coloca el **mejor candidato** en la matriz. Un estado en el que se haya completado una línea completa con solo **X** significa que el usuario ha ganado la partida y se representa con un 1. Un estado en el que se haya completado una solución por parte del algoritmo (o sea que hay una línea con **O** completa) se representa con un 2. Por último, si ya no hay espacios disponibles y ninguno de los jugadores ganó quiere decir que hay un empate, esto se representa con un 3.

El algoritmo voraz consta de 4 secciones o funciones principales:

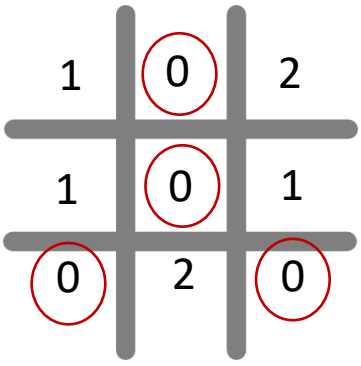
1- Conjunto de candidatos y selección
2- Viabilidad de los candidatos seleccionados
3- Objetivo
4- Solución

Ahora se va a explicar cada una de las secciones que forman parte del algoritmo;

1 - Conjunto de candidatos y selección:

Primero hay que definir qué es el conjunto de candidatos.

Conjunto de candidatos: En el algoritmo voraz se le llama conjunto de candidatos a aquellos elementos o posiciones de toda la matriz que tienen un 0, tal y como se puede ver en la figura de al lado.



En esta sección lo que se busca es encontrar todas las posiciones que están disponibles para más adelante poder evaluarlas y escoger a la que más nos puede **llevar a una solución** (entiéndase solución a una línea completa, de tamaño mayor o igual a 3 y con solamente 1's o solamente 2's).

Para poder encontrar los candidatos se debe recorre la matriz →((1 0 2) (1 0 1) (0 2 0)) y tomar solo los elementos que tengan un 0.

Para representar a cada uno de los candidatos seleccionados se va a utilizar la siguiente lista → (fila columna Plus) en donde:

- **fila:** La fila en la que se encuentra el candidato
- **columna:** La columna en la que se encuentra el candidato
- **Plus:** Este es un **valor de viabilidad** que se le asigna a cada candidato, entre más grande sea este valor significa que el candidato tiene más posibilidades de contribuir a una solución futura.

Se debe tener en cuenta que al inicio todos los candidatos seleccionados tienen un **Plus** igual a 0 ya que aún no se les ha asignado un **valor de viabilidad**.

Si tomamos a la matriz que tiene la figura anterior podemos decir que la salida que debería de tener la función de selección es → ((1 2 0) (2 2 0) (3 1 0) (3 3 0)).

2 - Viabilidad de los candidatos seleccionados:

La función de esta sección debe tomar a cada uno de los **candidatos seleccionados** y evaluarlos para poder asignarles un **Plus** que va a representar su **valor de viabilidad**.

Primero se va a introducir una nueva definición que llamaremos **vector solución**.

Un **vector solución** es una lista de posiciones de la matriz en donde se puede formar una línea (ya sea vertical, horizontal o diagonales). Un vector solución representa entonces una **solución** ya que si se completa una línea recta con los mismos valores (1's o 2's) en la matriz equivale a un **gane** de alguno de los jugadores (ya sea el algoritmo o el usuario).

Ahora, una lista de posiciones se puede considerar un **vector solución** si cumple con las siguientes dos condiciones:

1- El tamaño de la lista es mayor o igual a 3.

2- Sus elementos tienen valores de 0's y 2's (en caso de que sea un vector solución para el algoritmo) o tienen valores de 0's y 1's (en caso de que sea un vector solución para el usuario)

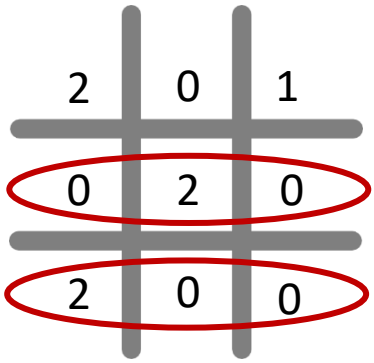
Una vez definido lo que es un vector solución, podemos definir lo que llamamos la **lista de vectores soluciones**. Tal y como se puede intuir, la lista de vectores soluciones es simplemente una lista que contiene elementos que son **vectores soluciones**.

En este algoritmo se requiere el uso de dos **listas de vectores soluciones**:

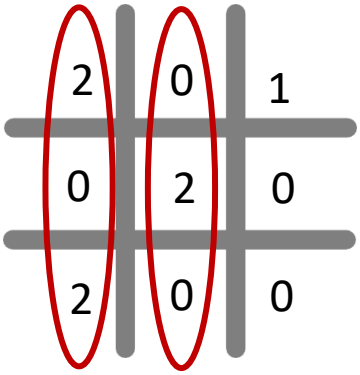
- **Vectores soluciones del algoritmo**: Esta lista va a contener todas las posibles soluciones que se pueden formar con valores de 2.

- **Vectores soluciones del usuario**: Esta lista va a contener todas las posibles soluciones que se pueden formar con valores de 1.

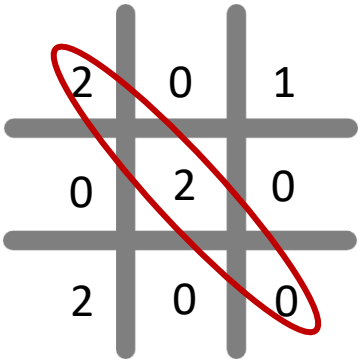
En la siguiente figura se puede ver cuáles serían los vectores soluciones del algoritmo (sus espacios se llenan con 2's) para una matriz 3 x 3 serían:



Vectores soluciones fila



Vectores soluciones columna



Vectores soluciones diagonales

Como se puede observar en las figuras, aquellas líneas que contengan uno o más valores del otro jugador (en el ejemplo los valores del otro jugador tienen un 1) no forman parte de los **vectores soluciones** ya que no es posible llegar a una **solución** completando esa línea.

Más adelante se dará un buen uso a los **vectores soluciones** que ya hemos definido.

Ahora se va a explicar el método que usa el algoritmo para darle **valores de viabilidad** a cada uno de los elementos que forma parte de los **candidatos seleccionados**.

Para poder evaluar la viabilidad vamos a usar un método que llamaremos **entrevista**.

La **entrevista** toma la lista de los candidatos y le asigna un **valor de viabilidad** a cada candidato seleccionado siguiendo los siguientes criterios de evaluación.

- **Plus + 1** si el candidato **es miembro** de un vector solución de la lista de vectores soluciones. Esto quiere decir si por ejemplo, el candidato es miembro de 4 vectores soluciones se le asigna (Plus + 4) 1 por cada intersección.

- **Plus + n** si el candidato es miembro de un vector solución y este vector tiene n valores colocados (2´s para el algoritmo y 1's para el usuario). Esto quiere decir, si un candidato es miembro de un vector y además ese vector solución contiene elementos con valores válidos para una solución, se le asigna (**Plus + n).**

- **Plus + 1000** si el candidato es un **killer**. Llamaremos **killer** a aquella posición en la matriz que falta para completar una línea y poder tener una línea completa. Le sumamos 1000 al Plus ya que un candidato que al elegirlo me dé un gane es el mejor candidato posible (recordemos que lo que se desea es ganar).

Luego de "entrevistar" a todos los **candidatos seleccionados** se va a tener una lista de **candidatos entrevistados** donde todos van a tener un **valor de viabilidad** o **Plus** que indica **qué tan bueno sería elegir esa posición**.

3 - Objetivo:

Una vez se tengan los **candidatos entrevistados** llega el momento de **escoger al mejor candidato** para colocar en la matriz.

La lista de candidatos es de tipo → ((1 2 4), (2 4 3)) lo que en este caso se lee de la siguiente manera: El primer elemento se encuentra en la fila 1, columna 2 y además tiene un Plus de 4, el segundo elemento se encuentra en la fila 2, columna 4 y tiene un Plus de 3.

En la lista de **candidatos entrevistados** podemos tener 3 posibles casos:

1- Hay un candidato que tiene el **Plus** más alto y además es mayor o igual a 1000. Esto indica que este candidato es un **killer**.

2- Hay un candidato que tiene el mayor **Plus** pero no es 1000. Esto indica que, aunque ese candidato no sea un **killer** es el mejor candidato para ese turno.

3- Hay dos o más candidatos que tienen el **Plus** más alto. Esto indica que esos candidatos son los mejores y tienen las mimas posibilidades de aportar a una **solución parcial o total**.

4- TODOS los candidatos tienen el mismo **valor de viabilidad** y este es igual a 0. Este caso es posible solo si los candidatos no son miembros de ningún **vector solución**, lo que significa que ya no hay manera de llegar a una solución (no hay líneas que se puedan completar para ganar).

Ahora, debido a que la lista de candidatos tiene elementos con distintos **valores de viabilidad** se procede a **ordenar la lista de candidatos entrevistados en orden descendente**.

Por ejemplo: podríamos tener una lista de candidatos como la siguiente:

- **((1 2 2) (3 2 5) (3 4 6) (4 3 2))**

Si ordenamos esta lista de candidatos en orden descendente vamos a tener:

- **((3 4 6) (3 2 5) (1 2 2) (4 3 2))**

En nuestra implementación usaremos el **algoritmo de ordenamiento Quick Sort** para poder ordenar la **lista de candidatos entrevistados**.

Una vez ordenada la lista de candidatos entrevistados. Se debe **escoger al mejor candidato de la lista**. Pero para tomar esa decisión se deben seguir las siguientes reglas:

1- Si un candidato tiene un **Plus mayor o igual a 1000** significa que es un **killer**. Entonces se escoge ese candidato ya que si se coloca en la matriz se podrá llegar a una **solución completa** o viéndolo visualmente, se formará una línea completa con 2's.

2- No tenemos ningún **killer** pero el jugador (el que tiene 1's en sus posiciones) tiene un **killer**. Esto significa que si el jugador marca esa posición (el killer) en el siguiente turno va a ganar. Por lo tanto, se debe escoger el killer del jugador para tapar su línea y evitar que gane.

3- No tenemos ningún **killer**,el jugador tampoco (a ninguno le falta una posición para ganar) y el primer elemento de la lista de candidatos tiene un **Plus** mayor que el **Plus** del segundo candidato. Ya que tenemos la lista de candidatos ordenados de manera descendente, esta condición implica que el primer candidato de la lista es el mejor, por lo tanto, se escoge ese candidato.

4- No tengo ningún **killer**, el jugador tampoco y el primer elemento de la lista tiene un **Plus** igual que el segundo. Este caso indica que existen dos o más candidatos que se pueden considerar como **los mejores**. Entonces se procede a elegir **cualquiera** de esos candidatos.

Cuando ya se tenga al **mejor candidato**, lo que procede es **actualizar la matriz del turno y actualizar la lista de vectores soluciones**.

Para actualizar la matriz solo se debe tomar al mejor candidato → (fila columna) y colocarlo en la posición correspondiente.

Para actualizar los vectores soluciones solo se debe tomar al mejor candidato → (fila columna) y recorrer la lista de vectores soluciones. Se debe colocar el elemento en todos los vectores a los que pertenece o interseca.

4 - Solución:

Ahora se debe verificar si el candidato que se escogió anteriormente ha contribuido a llegar a una **solución** (al colocarlo en la matriz se ha completado una línea y por ende se ha ganado).

Hasta este punto de la ejecución del algoritmo, ya se tiene **al mejor candidato seleccionado → (fila columna), una matriz actualizada con el mejor candidato colocado y una lista de vectores soluciones actualizada**.

Para poder verificar si ya existe una **solución** solo basta con revisar si la lista de los **vectores soluciones actualizada** tiene algún **vector solución** que tenga valores de 2 en todos sus elementos.

Un ejemplo de una **lista de vectores soluciones actualizada** sería la siguiente:

- ‘(((1 1 0) (1 2 2) (1 3 2)) ((1 1 2) (2 2 2) (3 3 2)))

Como se observa en la lista anterior, el primer vector solución → ‘((1 1 0) (1 2 2) (1 3 2)) es un **killer** ya que solo le falta completar la posición (1 1) para tener una **solución** y el segundo vector → ((1 1 2) (2 2 2) (3 3 2)) es una **solución** ya que todos sus elementos tienen valor de 2, lo que significa que **esa línea está completa**.

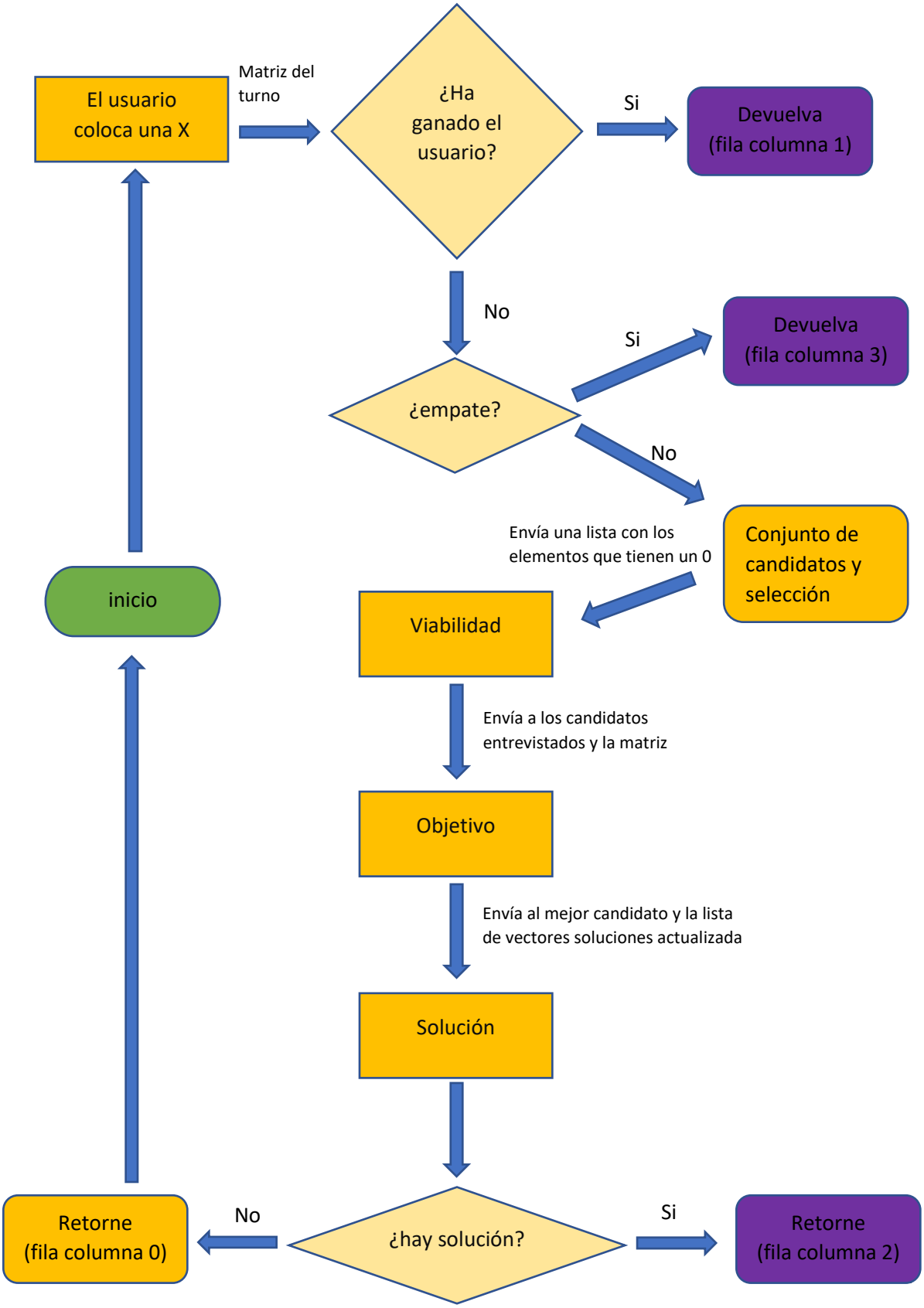
Como se vió en la primer figura, el **algoritmo voraz** debe devolver un elemento de tipo **(fila columna resultado)** al final de su ejecución. Si se ha llegado hasta este punto del algoritmo, ya se está en la capacidad de **brindar una salida**. Para eso se deben validar las siguientes condiciones:

1- Si se encontró una **solución** en la lista de vectores soluciones, significa que el algoritmo logró formar una línea completa y pudo ganar. Entonces se da como resultado → **(fila columna 2)**.

2- Si no se encontró ninguna solución significa que aún no se ha ganado. Entonces se da como resultado → **(fila columna 0)**

En ambas salidas **fila** es la fila del **mejor candidato encontrado** en la sección objetivo, **columna** es la columna del **mejor candidato encontrado** en la sección objetivo y el último elemento es lo que llamamos **resultado**, el cuál toma el valor de 0, 1, 2 y 3 (esto se explicó al inicio del documento).

Ahora que se han explicado las secciones del algoritmo voraz que se implementó en la tarea, se puede entender el siguiente diagrama que representa la **ejecución del algoritmo en cada turno**.


El usuario coloca una X
Matriz del turno
¿Ha ganado el usuario?
Si
Devuelva (fila columna 1)
No
¿empate?
Si
Devuelva (fila columna 3)
No
Conjunto de candidatos y selección
Envía una lista con los elementos que tienen un 0
Viabilidad
Envía a los candidatos entrevistados y la matriz
Objetivo
Envía al mejor candidato y la lista de vectores soluciones actualizada
Solución
¿hay solución?
Si
Retorne (fila columna 2)
No
Retorne (fila columna 0)
inicio

Descripción de todas las funciones implementadas

Funciones para el Algoritmo Goloso

Conjunto de Candidatos y Selección

Nombre:	seleccionar-candidatos
Descripción:	El objetivo de esta función es insertar en una lista aquellos elementos que estén formados por un 0 eso quiere decir que los elementos están disponibles para ser elegidos. Los elementos que conformarán la lista serán de la forma ---> (fila columna 0) donde el 0 equivale al valor de plus (peso de viabilidad, que es 0 porque aún no se ha evaluado).
Entrada	La matriz del turno actual
Salida	Una lista con los elementos de la matriz que están disponibles
Tomado de:	Creación propia

Nombre:	seleccionar_candidatos_aux
Descripción:	Elige solo a los elementos que tienen un 0 y los guarda en una lista con su fila y columna respectiva.
Entrada	matriz fila columna candidatos
Salida	NA
Tomado de:	Creación propia

Viabilidad

Nombre:	vectores_soluciones
Descripción:	Esta función forma una lista con todas las posibles soluciones (líneas con tamaño igual o mayor a 3) que se pueden obtener de una matriz (verticales, horizontales, diagonales y diagonales inversas). Se debe tomar en cuenta el filtro de los vectores ya que hay líneas que son de tamaño correcto, pero tienen uno o varios elementos con un valor del otro jugador, esto quiere decir que esa línea no es una solución parcial.

Entrada	matriz, valor Una matrix mxn exacta (o sea que los elementos sean de tipo (fila columna plus)) y el valor del jugador (1 si se desean todas las posibles soluciones del usuario y un 2 si se desean todas las posibles soluciones del algoritmo)
Salida	Una lista con todos los vectores soluciones posibles para el valor brindado
Tomado de:	Creación propia

Nombre:	agregar_diagonal_inversa
Descripción:	Función con la cual se agregan las diagonales inversas (\)
Entrada	vectores_total valor
Salida	Lista de diagonales inversas
Tomado de:	Creación propia

Nombre:	agregar_diagonal
Descripción:	Se agregan las diagonales (/)
Entrada	matriz_exacta vectores_total valor
Salida	Lista de diagonales
Tomado de:	Creación propia

Nombre:	agregar_columna
Descripción:	Se agregan las columnas(|)
Entrada	vectores_total valor
Salida	Lista de columnas
Tomado de:	Creación propia

Nombre:	agregar_fila
Descripción:	Se agregan las filas (--) y se pasan todos los vectores por el filtro
Entrada	vectores_total valor
Salida	Lista de filas
Tomado de:	Creación propia

Nombre:	entrevista
Descripción:	Esta función se va a encargar de entrevistar/evaluar a cada uno de los candidatos y les va a dar un valor de viabilidad -->(plus) , entre más alto sea el plus, más viable es para ser elegido como la mejor posición Los criterios de evaluación son los siguientes: + 1 si un candidato iterseca (es miembro) con un vector solución (fila, columna, diagonal o diagonal inversa) + n si un candidato interseca con un vector solución y ese vector contiene n posiciones ocupadas por un 2 + 1000 si un candidato interseca con un vector solución y a ese vector solo le falta una posición para formar la línea completa.
Entrada	Candidatos y vectores_solucion Una lista con todos los candidatos seleccionados y la lista de todos los vectores solución
Salida	una lista con todos los candidatos, pero con su Plus evaluado, o sea con un valor de viabilidad
Tomado de:	Creación propia

Nombre:	entrevista_aux
Descripción:	Esta es la función auxiliar de entrevista, su trabajo es tomar a todos y cada uno de los candidatos seleccionados y enviarlos a la función Plus junto con la lista de vectores soluciones para poder darles un valor de viabilidad.
Entrada	vectores-solucion, candidatos, potenciales
Salida	Lista de candidatos y lista de vectores
Tomado de:	Creación propia

Nombre:	Plus
Descripción:	Esta función se encarga de recibir un candidato junto con una lista de vectores soluciones. Toma en cuenta los criterios de evaluación citados anteriormente y le da un valor a cada candidato que reciba.
Entrada	candidato vectores_solucion suma
Salida	Lista con los valores de los candidatos
Tomado de:	Creación propia

Nombre:	Boost
Descripción:	Esta función se usa cuando un vector tiene varios elementos con solamente 2's y 0's o cuando tiene varios elementos con solamente 1's 0's. Se encarga de hacer un conteo de los 2's o 1's que tiene el vector
Entrada	vector, valor, suma Un vector y un número que siempre es 0 (la idea es sumarle 1 por cada coincidencia que encuentre)
Salida	la cantidad de coincidencias que hay en el vector con respecto al valor dado
Tomado de:	Creación propia

Nombre:	killer?

Descripción:	Esta función se encarga de verificar si un candidato es el último elemento que le falta a un vector para formar una línea completa en la matriz
Entrada	candidato, vector, valor un candidato, un vector o lista y el valor del jugador (1 si quiero verificar el killer del usuario y 2 si quiero verificar el killer del algoritmo)
Salida	#t si efectivamente es un killer y #f en caso contrario
Tomado de:	Creación propia

Nombre:	killer_enemigo?
Descripción:	Esta función verifica si el enemigo (entiendase el usuario que está jugando) tiene un killer en la matriz Es importante saber si el enemigo cuenta con esta posibilidad ya muy posiblemente se deba tapar esa posición
Entrada	La lista de todos los candidatos y los vectores soluciones del usuario
Salida	
Tomado de:	Creación propia

Nombre:	killer_vectores?
Descripción:	Función auxiliar de killer_enemigo, verifica la posibilidad de killers pero para un candidato.
Entrada	candidato vectores valor
Salida	NA
Tomado de:	Creación propia

Nombre:	dame_killer_enemigo
Descripción:	Esta función se encarga de retornar el la posición de un posible killer del usuario
Entrada	la lista de candidatos evaluados y la lista de los vectores soluciones del usuario
Salida	La posición del elemento killer del enemigo.
Tomado de:	Creación propia

Nombre:	dame_killer_enemigo_aux
Descripción:	Busca al posible killer tomando solo a un candidato de toda la lista de candidatos
Entrada	Candidato,vectores, valor
Salida	NA
Tomado de:	Creación propia

Objetivo

Nombre:	el-elegido
Descripción:	Esta función se encarga de seleccionar al mejor candidato para formar una línea completa o parcial en la matriz, debe existir una lista con todos los candidatos con un valor de viabilidad o Plus asociado. Para tomar la decisión del mejor candidato se basa en los siguientes criterios: 1- Existe un killer en la lista de candidatos: si un elemento de la lista de candidatos es un killer, significa que si selecciono ese elemento estaría completando una línea en la matriz a mi favor. Entonces se selecciona ese elemento inmediatamente. 2- No hay killer, pero hay un elemento que tiene el mayor Plus: Estas condiciones significan que el elemento que tiene mayor

	Plus tiene muchas posibilidades de contribuir a una solución futura. Entonces se selecciona ese elemento. 3- No hay killer y hay más de un elemento que tiene el mayor Plus: Esto significa que existen varios elementos que tienen las mismas posibilidades de contribuir a una solución parcial, por lo que no importa cuál de ellos se elija, voy a tener un resultado similar. Entonces se elige al candidato aleatoriamente mediante una función.
Entrada	La lista de los candidatos entrevistados y ordenados en ordenados en orden descendente y la matriz del turno
Salida	La mejor posición para el turno
Tomado de:	Creación propia

Nombre:	elegido-aleatorio
Descripción:	Esta se encarga de seleccionar un elemento al azar
Entrada	La lista de candidatos entrevistados y un par de posiciones que representan el inicio de la lista de candidatos y una posición random
Salida	Un elemento elegido aleatoriamente
Tomado de:	Creación propia

Nombre:	Actualizar-matriz
Descripción:	Esta función se encarga de colocar al mejor candidato seleccionado en la matriz del turno
Entrada	El mejor candidato, el valor del jugador (1 o 2) y la matriz del turno
Salida	La matriz actualizada con el nuevo valor
Tomado de:	Creación propia

Nombre:	actualizar-vectores-solucion
Descripción:	Esta función se encarga de colocar al mejor candidato dentro de los vectores soluciones de la matriz (esto si el elemento forma parte de alguno o algunos de los vectores soluciones)
Entrada	El mejor candidato, la lista de los vectores soluciones y una lista vacía para guardar los vectores actualizados
Salida	una lista con los vectores soluciones actualizados
Tomado de:	Creación propia

Nombre:	actualiza-vector
Descripción:	Función auxiliar para actualizar los vectores soluciones, esta función actualiza a los vectores individuales
Entrada	elegido vector final
Salida	NA
Tomado de:	Creación propia

Solución

Nombre:	solucion_jugador?
Descripción:	Esta función verifica si el jugador ya tiene una línea completa formada en la matriz
Entrada	la matriz del turno y los vectores soluciones del jugador
Salida	#t si el jugador ya ganó y #f en caso contrario
Tomado de:	Creación propia

Nombre:	solucion?
Descripción:	Esta función valida si el algoritmo ya formó una línea completa después de colocar al mejor candidato en la matriz del turno. Además, debe

	retornar la mejor posición junto con un estado del juego (1 2 3)
Entrada	los vectores soluciones actualizados y al mejor candidato
Salida	una lista del tipo ---> (fila columna estado)
Tomado de:	Creación propia

Nombre:	Empate?
Descripción:	Esta función determina si en la matriz ya no hay espacios vacíos, lo que significa que hay un empate.
Entrada	la matriz del turno
Salida	#t si hay empate y #f en caso contrario
Tomado de:	Creación propia

Nombre:	Fila-zero?
Descripción:	Función auxiliar para ver si hay empate, busca coincidencias de 0's por fila
Entrada	Recibe la fila que se va analizar
Salida	True o false en caso de que este vacia true
Tomado de:	Creación propia

Pruebas y Definiciones Finales

Nombre:	Algoritmo-codicioso
Descripción:	Esta función relaciona todas las secciones para retornar la mejor posición a partir de la matriz del turno
Entrada	La matriz actual
Salida	
Tomado de:	Creación propia

Nombre:	selección

Descripción:	Esta función toma la matriz del turno y coloca a todas las posiciones disponibles en una lista de candidatos Además, a cada uno de los candidatos seleccionados, les da un valor de Plus de 0, ya que aún no han sido entrevistados.
Entrada	matriz
Salida	Cuando ya se haya completado el proceso de selección se le envía la lista de candidatos seleccionados y la matriz a la función de viabilidad para que los evalúe.
Tomado de:	Creación propia

Nombre:	viabilidad
Descripción:	Esta función toma a la matriz del turno y la lista de candidatos seleccionados y se va a encargar de evaluar a cada uno de los candidatos para darles un valor de Plus.
Entrada	Matriz, candidatos,_seleccionados
Salida	Cuando ya se haya finalizado el proceso de viabilidad se le envía la matriz, la lista de los candidatos entrevistados y la lista de los vectores soluciones a la función objetivo.
Tomado de:	Creación propia

Nombre:	objetivo
Descripción:	Esta función toma la lista de candidatos entrevistados la lista de los vectores soluciones para elegir al candidato que venga con el mayor valor de Plus y así actualizar la matriz y los vectores soluciones.

Entrada	matriz candidatos_entrevistados vectores_solucion
Salida	Cuando haya finalizado el proceso envía la matriz actualizada, los vectores actualizados y el candidato elegido, a la
Tomado de:	Creación propia

Nombre:	objetivo_aux
Descripción:	función auxiliar de objetivo
Entrada	matriz elegido vectores_solucion
Salida	NA
Tomado de:	Creación propia

Funciones en la Interfaz

Nombre:	TTT
Descripción:	función que inicializa el juego
Entrada	Filas, columnas
Salida	NA
Tomado de:	Creación propia

Nombre:	juego
Descripción:	Ciclo principal del juego, aquí se van a ejecutar todos los turnos, siempre y cuando el usuario Si no lo ha hecho, el programa se queda esperando a que le dé click izquierdo haya presionado clcik izquierdo.
Entrada	Filas, columnas
Salida	No aplica
Tomado de:	Creación propia

Nombre:	inicializar_partida
Descripción:	Función que se llama una vez se haya acabado una partida Se encarga de inicializar las definiciones de matrices y las capas
Entrada	Filas, columnas
Salida	No aplica
Tomado de:	Creación propia

Nombre:	margenes_fila
Descripción:	Función que recibe el número de filas y retorna la posición los márgenes de esa fila como una lista debe devolver (margen-inicial margen-final)
Entrada	Filas, columnas
Salida	margen-inicial margen final
Tomado de:	Creación propia

Nombre:	margenes_columna
Descripción:	Función que recibe el número de filas y retorna la posición los márgenes de esa fila como una lista debe devolver (margenes)
Entrada	Filas, columnas
Salida	margenes
Tomado de:	Creación propia

Nombre:	en_matriz?
Descripción:	Función que valida si la posición elegida con el mouse se encuentra dentro de los márgenes de la matriz que se creó
Entrada	posx, posy Filas, columnas
Salida	true si está dentro de los márgenes, false si no lo esta

Tomado de:	Creación propia

Nombre:	modificar_tablero_grafico
Descripción:	Función que recibe una posición X y Y Se debe colocar una X en dicha posición
Entrada	posx, posy tipo_figura
Salida	No aplica
Tomado de:	Creación propia

Nombre:	localizar_columna
Descripción:	Función que recibe una posición en X y valida en cuál columna se encuentra dentro del tablero devuelve la posición X que se debe usar para dibujar en dicha columna
Entrada	posx, margenes
Salida	La posicion X
Tomado de:	Creación propia

Nombre:	numero_columna
Descripción:	Función que recibe una posición en X y valida en cuál columna se encuetra dentro del tablero devuelve el número de columna en el que se encuentra posX
Entrada	posx, margenes
Salida	el número de columna donde está la posición X
Tomado de:	Creación propia

Nombre:	localizar_fila
Descripción:	Función que recibe una posición en Y validando en cuál fila se encuentra dentro

	del tablero devuelve la posición Y que se debe usar para dibujar en dicha fila
Entrada	posY, margenes
Salida	La posicion Y donde se quiere dibujar
Tomado de:	Creación propia

Nombre:	numero_fila
Descripción:	Función que recibe una posición en Y y valida en cuál fila se encuentra dentro del tablero devuelve el número de fila en el que se encuentra posY
Entrada	posY, margenes
Salida	La fila donde se encuentra posY
Tomado de:	Creación propia

Nombre:	numero_fila
Descripción:	Función que recibe el número de fila y retorna la posición en Y correspondiente para poder dibujar
Entrada	fila, margenes
Salida	La posición donde dibujar Y
Tomado de:	Creación propia

Nombre:	dame_pos_c
Descripción:	Función que recibe el número de columna y retorna la posición en X correspondiente para poder dibujar
Entrada	fila, columna margenes
Salida	La posicion en X
Tomado de:	Creación propia

Nombre:	crear_tablero
Descripción:	Función que dado el numero de filas y columnas dibuja el tablero correspondiente
Entrada	fila, columna
Salida	La posicion en X
Tomado de:	Creación propia

Nombre:	desplegar_bienvenida
Descripción:	Función que muestra la pantalla de bienvenida durante 5 segundos
Entrada	NA
Salida	NA
Tomado de:	Creación propia

Nombre:	conclusion
Descripción:	Función que despliega una imagen en caso de llegar a un gane un 2 equivale a una derrota
Entrada	Resultado
Salida	NA
Tomado de:	Creación propia

Nombre:	colocar_figura
Descripción:	Función que recibe una un valor (X o O) y una posición Dibuja una figura del tipo que se le pase por parámetro en la posición dada
Entrada	tipo posX posY
Salida	NA
Tomado de:	Creación propia

Nombre:	mensaje_alerta
Descripción:	Función que muestra un mensaje de alerta recibe una mensaje de tipo string
Entrada	Mensaje tiempo
Salida	NA
Tomado de:	Creación propia

Nombre:	construir_matriz
Descripción:	Función que se encarga de contar las columnas
Entrada	Mensaje tiempo
Salida	NA
Tomado de:	Creación propia

Nombre:	crear_fila
Descripción:	Función que se encarga de contar las filas
Entrada	columnas contador lista
Salida	Cantidad filas
Tomado de:	Creación propia

Nombre:	modificar_matriz_logica
Descripción:	Función que recibe el número de fila, columna y un valor devuelve una matriz con la nueva posición actualizada
Entrada	matriz fila columna valor
Salida	matriz (con la posicion que se cambio)
Tomado de:	Creación propia

Nombre:	colocar

Descripción:	Función de colocar recibe una matriz del tipo ((1 1 0) (1 2 2) (1 3 0))
Entrada	fila columna contador valor matriz matriz_final
Salida	posicion donde colocar
Tomado de:	Creación propia

Nombre:	Colocar-aux
Descripción:	Función auxiliar de colocar recibe una matriz del tipo ((1 1 0) (1 2 2) (1 3 0))
Entrada	lista columna fila contador valor lista_final
Salida	lista donde colocar
Tomado de:	Creación propia

Nombre:	disponible?
Descripción:	Función que recibe el número de fila y columna me dice si ese espacio tiene un 0, o sea si está disponible
Entrada	fila columna matriz
Salida	True si esta disponible, false si no lo esta
Tomado de:	Creación propia

Nombre:	buscar_fila
Descripción:	Función que recibe el numero de fila y columna me da la fila a la que quiero acceder
Entrada	fila columna matriz
Salida	NA
Tomado de:	Creación propia

Nombre:	disponible?
Descripción:	Función de colocar recibe una matriz del tipo ((1 1 0) (1 2 2) (1 3 0))
Entrada	fila columna contador valor matriz matriz_final
Salida	posicion donde colocar
Tomado de:	Creación propia

Nombre:	buscar_columna
Descripción:	Función que me da la columna a la que quiero acceder
Entrada	columna contador fila
Salida	True si esta disponible, false si no lo esta
Tomado de:	Creación propia

Descripción de la ejemplificación de las estructuras de datos desarrolladas

En esta sección se van a describir todas las estructuras de datos desarrolladas y la representación que se le debe asociar a cada una.

Las estructuras de datos que se implementaron en esta tarea son las siguientes:

1- Estructuras desarrolladas en la lógica del juego:
   - Matriz lógica del algoritmo
   - Matriz exacta
   - Conjunto de candidatos
   - Vector solución
   - Lista de vectores soluciones
   - Output del algoritmo voraz
2- Estructuras desarrolladas en la interfaz del juego:
   - Márgenes filas
   - Márgenes columnas
   - Matriz lógica de la interfaz

1: Estructuras desarrolladas en la lógica del juego:

Primero se van a describir las estructuras que se usaron en los módulos de la lógica.

Matriz lógica del algoritmo

Esta es la representación del tablero que se muestra en el juego. Esta estructura tiene la siguiente simbología:

- Un 0 → equivale a un espacio vacío en el tablero.
- Un 1 → equivale a una **X** en el tablero.
- Un 2 → equivale a un **O** en el tablero.

En código se va a representar de la siguiente manera:

- **‘((0 0 0) (0 1 2) (1 2 0))**

Matriz exacta

Una matriz lógica del algoritmo se puede ver como una matriz exacta. Una matriz exacta a diferencia de una lógica no muestra solamente el valor que tiene cada elemento de la matriz, sino que además muestra sus posiciones respectivas (o sea sus filas y sus columnas).

Por ejemplo: se va a representar la matriz lógica del ejemplo anterior como una matriz exacta.

- **‘(((1 1 0) (1 2 0) (1 3 0)) ((2 1 0) (2 2 1) (2 3 2)) ((3 1 1) (3 2 2) (3 3 0)))**

Conjunto de candidatos

Es una lista que contiene a todos los valores de la matriz que se encuentran disponibles, dicho de otra forma, aquellos que tiene un valor de 0. Cada uno de los candidatos se va a representar de la siguiente manera:

- (fila columna Plus)

En donde **fila** es el número de fila en la que se encuentra, **columna** es el número de columna en la que se encuentra y **Plus** es un valor de viabilidad que se usa para simbolizar qué tan bueno es cierto candidato si se elige.

Por ejemplo: se puede tener una lista que representa al conjunto de candidatos de cierta matriz.

- **‘((1 1 0) (3 4 0) (2 5 0) (2 1 0))**

Al inicio todos los candidatos tienen Plus igual a 0 porque aún no se han evaluado en la función **entrevista** de la sección de viabilidad.

Vector Solución

Un vector solución representa una línea total o parcial en la matriz. En el código se representa a un vector solución como una lista que contiene los elementos que forman una línea en la matriz.

Cada elemento del vector solución se representa de la siguiente manera:

- (fila columna valor)

Por ejemplo: un vector solución de cierta matriz se representa de la siguiente manera.

- **‘((1 1 0) (1 2 2) (1 3 0))**

Lista de vectores soluciones:

Una lista de vectores soluciones es simplemente una lista que contiene todos los vectores soluciones de cualquier matriz. Toda matriz puede tener vectores soluciones horizontales, verticales o diagonales o de todos los tipos combinados.

Por ejemplo: una lista de los vectores soluciones de cierta matriz se puede representar de la siguiente manera.

- **'(((1 1 0) (1 2 0) (1 3 0)) ((1 1 2) (2 2 0) (3 3 2)))**

Output del algoritmo voraz

El algoritmo voraz que se implementó debe de retornar una lista que contenga las posiciones del mejor candidato escogido y además un estado del juego.

Para poder retornar esta información se requirió de la siguiente lista:

- (fila columna resultado)

En donde **fila** es la fila en la que se encuentra al candidato escogido por el algoritmo, **columna** es la columna en la que se encuentra al candidato escogido por el algoritmo y **resultado** es el estado del juego.

El valor de **resultado** puede ser:

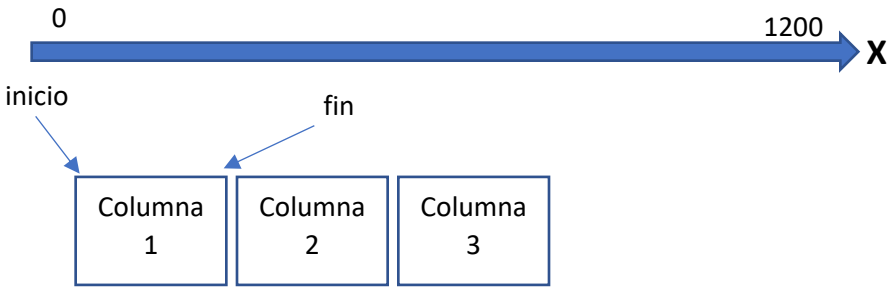
- Un 0 si la partida no ha terminado cuando se coloque el nuevo valor
- Un 1 si el usuario ganó
- Un 2 si el algoritmo logró ganar
- Un 3 si hay un empate

2: Estructuras desarrolladas en la interfaz del juego:

Luego, se van a describir las estructuras que se usaron en los módulos de la lógica

Márgenes columnas

Esta es una representación de los pixeles que demarcan los límites de cada columna en la ventana del juego (1200 x 680).



Cada elemento se representa de la siguiente manera → **(columna inicio fin)**

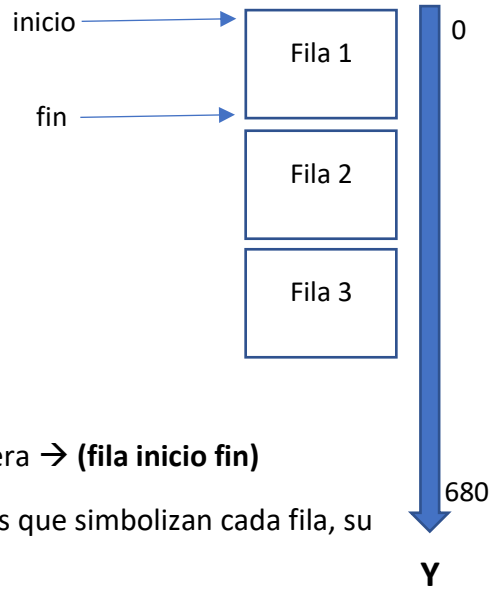
Márgenes columnas es una lista que contiene elementos que simbolizan cada columna, su inicio en la ventana y su fin en la ventana.

En esta tarea la lista de márgenes columnas siempre va a ser la misma y es la siguiente:

- **''((1 270 330) (2 330 390) (3 390 450) (4 450 510) (5 510 570) (6 570 630) (7 630 690) (8 690 750) (9 750 810) (10 810 870)))**

Márgenes filas

Esta es una representación de los pixeles que demarcan los límites de cada fila en la ventana del juego (1200 x 680).



Cada elemento se representa de la siguiente manera → **(fila inicio fin)**

Márgenes filas es una lista que contiene elementos que simbolizan cada fila, su inicio en la ventana y su fin en la ventana.

En esta tarea la lista de márgenes filas siempre va a ser la misma y es la siguiente:

- **'((1 10 70) (2 70 130) (3 130 190) (4 190 250) (5 250 310) (6 310 370) (7 370 430) (8 430 490) (9 490 550) (10 550 610)))**

Matriz lógica en la interfaz

Se usa para guardar los estados del tablero y también para hacer validaciones de posiciones o ver si un espacio se encuentra disponible para poner una X.

Su representación es la misma que la de la **matriz exacta**, solo que esta se va a usar en la interfaz.

Problemas sin solución

Todos los problemas fueron solucionados.

Plan de Actividades realizadas por estudiante				
Descripción de la tarea	Tiempo estimado	Responsable a cargo	Estado	Fecha de entrega
Investigar sobre creación de matrices	2 horas	Kevin	Entregado	19/06
Investigar sobre operaciones de matrices	2.5 horas	Kevin	Entregado	19/06
Investigar sobre librerías de la interfaz gráfica	2 horas	Hazel	Entregado	20/06
Creación del tablero	2 horas	Hazel	Entregado	22/06
Creación del tableros generalizado	5 horas	Hazel	Entregado	23/06
Implementar la función de selección	7 Horas	Kevin	Entregado	23/06
Crear un documento visual y en pseudocódigo que permita mostrar el comportamiento del algoritmo goloso, esto para tener una idea clara antes de programar el algoritmo	4 Horas	Kevin	Entregado	23/06
Modificar la función de selección (agregar el plus inicializado en 0)	10 Minutos	Kevin	Entregado	25/06
Crear funciones que reciban la matrix mxn y retorne TODOS los vectores soluciones, además que descarten los que tienen 1´s	6 Horas	Kevin	Entregado	25/06
Crear función que actualice la lista de candidatos validando las intersecciones	5 Horas	Kevin	Entregado	26/06
Crear función que se encargue de actualizar la lista de candidatos y los vectores solución	2 Horas	Kevin	Entregado	26/06
Crear la función objetivo	1 Hora	Kevin	Entregado	26/06
Crear la función solución	2 Horas	Kevin	Entregado	27/06
Realizar las pruebas de la lógica y los ajustes necesarios	4 Horas	Kevin	Entregado	28/06
Crear la funcion que inicializa el juego	1 Hora	Hazel	Entregado	23/06

creación del ciclo del juego los turnos	3 Horas	Hazel	Entregado	25/06
Validaciones de la matriz	1 Hora	Hazel	Entregado	25/06
Colocar ficha en la matriz	2 Horas	Hazel	Entregado	26/06
Actualizar matriz	3 Horas	Hazel	Entregado	27/06
Ventana Bienvenida	1 Hora	Hazel	Entregado	27/06
Resultados (gana o pierde)	1 Hora	Hazel	Entregado	28/06
Unir el algoritmo con la interfaz	5 Horas	Kevin y Hazel	Entregado	28/06
Documentos necesarios	12 Horas	Kevin y Hazel	Entregado	29/06
Presentacion	8 Horas	Kevin y Hazel	Entregado	30/06

Problemas encontrados

La función dame_ diagonales no funciona para matrices que tengan más de 5 columnas:

Descripción: En la función dame_diagonales se usan dos pivotes para poder señalar las líneas que se deben extraer de la matriz, el primer pivote es el inicio de la línea diagonal y el segundo pivote es el final de la diagonal. La función devuelve una lista de diagonales correctas cuando recibe matrices que tengan menos de 5 columnas. Si recibe una que tenga más columnas devuelve listas que no coinciden con las diagonales deseadas.

Intentos de solución sin éxito: 0

Solución encontrada: Se cambió el punto de vista de los pivotes. En la versión anterior de la función se usaba un contador para aumentar el valor del segundo pivote, esto porque al aumentar el valor podemos tomar la siguiente diagonal de la matriz, sin embargo, el valor del primer pivote nunca se aumentaba, siempre era 1. Funcionaba bien para aquellas matrices que tengan 4 columnas o menos, pero no funcionaba en los otros casos.

La solución fue aumentar el valor del primer pivote cada vez que estuviera en el último elemento de la primea columna. Luego se eliminaba la columna y el primer pivote quedaba en la segunda, y así para n columnas de la matriz.

Recomendaciones: No se recomienda nada en especial.

Conclusiones: Es de gran utilidad tener en cuenta todos los casos posibles a la hora de programar una función.

Bibliografía consultada: No aplica.

No se ha logrado dibujar matrices mxn en la ventana del juego:

Descripción: En la interfaz gráfica del juego se están usando botones para representar cada una de las posiciones de la matriz. Se pueden dibujar matrices mxn, pero cada vez que se desee cambiar el numero de filas o columnas de la matriz se tiene que cambiar el código para agregar o quitar botones. No se ha podido implementar una función que dado el número de filas y columnas dibuje la matriz en la pantalla

Intentos de solución sin éxito: 0

Solución encontrada: Se cambió la representación y programación de las matrices en la ventana. Se decidió usar líneas horizontales y verticales para dibujar la matriz en la ventana. Por un lado, si se usan líneas para dibujar la matriz se va a ver como realmente se ve el juego de Tic Tac Toe en la vida real (papel y lápiz) y por el otro, dibujar líneas rectas en la ventana es más sencillo que colocar botones. Usando líneas se puede simplificar la

programación de las matrices. Por ejemplo, si el usuario pide una matriz 4x5, solo se deben dibujar 3 líneas horizontales y 4 líneas verticales.

Recomendaciones: Se puede entender mejor la idea dibujando las matrices en un papel.

Conclusiones: A veces las soluciones simples son las mejores para resolver problemas complicados.

Bibliografía consultada: No aplica.

No se logra visualizar las victorias de los jugadores en la ventana:

Descripción: Cuando se está ejecutando el juego y alguno de los dos jugadores (el usuario o el algoritmo voraz) logra ganar o el usuario pierde, es casi imposible ver la línea que formó porque cuando se detecta un final de partida se muestra inmediatamente una ventana de conclusión (game over o the winner).

Intentos de solución sin éxito: 0

Solución encontrada: Solo bastó con agregar un (sleep) para que se muestre la matriz junto con las posiciones marcadas por unos segundos más y luego se muestra la ventana de conclusión. Ahora se puede ver perfectamente cuál fue la línea que se completó.

Recomendaciones: No hay ninguna en especial.

Conclusiones: Las interfaces graficas deben ser amigables con el usuario y mostrar todos los datos relevantes.

Bibliografía consultada: No aplica.

Conclusiones

Se implemento el juego Tic Tac Toe utilizando el paradigma de programación funcional demostrando la rectificación del conocimiento sobre el mismo.

Se realizo la aplicación utilizando listas según las normas del paradigma funcional presentando las mismas como estructuras de datos.

Se implementaron las operaciones para listas según el paradigma de programación funcional de esta forma se emplearon según el control de datos del lenguaje LISP.

Recomendaciones

Localizar la mejor manera de realizar la interfaz y con las correctas librerías, debido a que primeramente se realizó la matriz con botones, pero de esta manera se complicó mucho teniendo que invertir tiempo en la reestructuración tanto de bibliotecas como de la forma de creación de la matriz.

Bibliografía Consultada

[1]"Graphics: Legacy Library", *Docs.racket-lang.org*, 2020. [Online]. Available: https://docs.racket-lang.org/graphics/index.html.