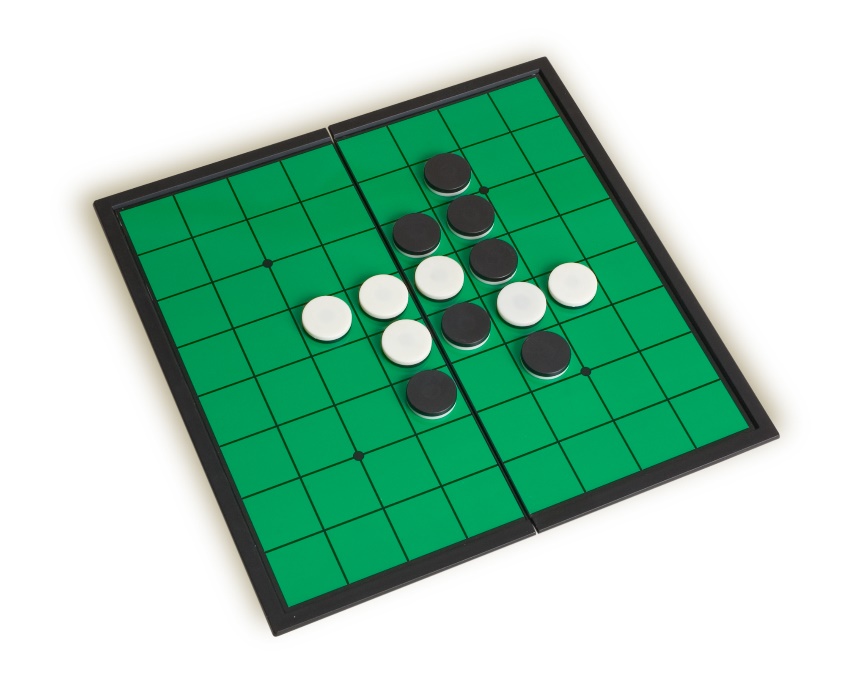
**Conclusiones proyecto PMD Infinireversi**

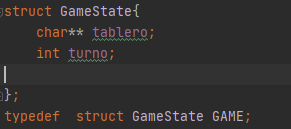
Carlos Andrés Páez Aguilar

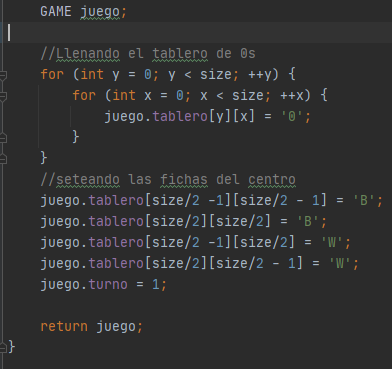
Oscar Javier Hernández Salamanca

A mediados de este semestre, la materia de PMD propuso el tan esperado proyecto final, con varias opciones para el salón, la opción por la que nosotros optamos se llama infinireversi, el juego consistía en el clásico reversi, un juego sencillo de mesa para 2 jugadores. La dificultad del juego es baja pero su implementación represento todo un reto para nosotros.



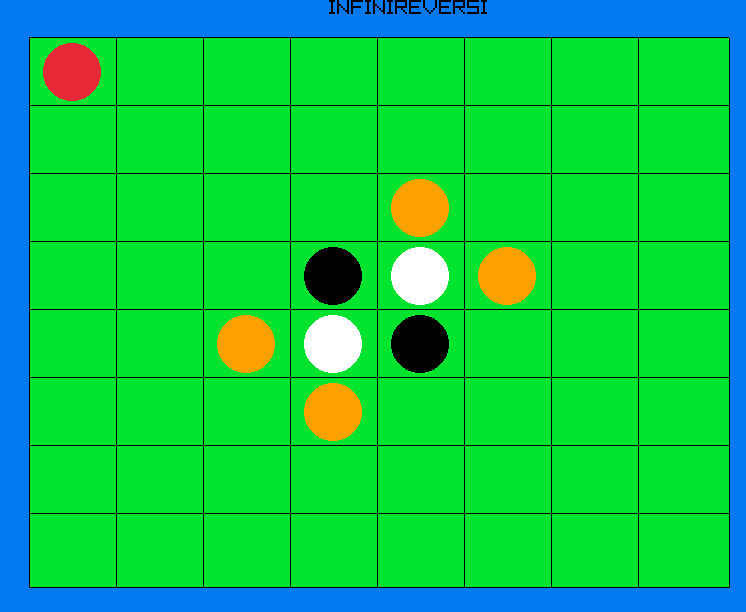
Desde un inicio pensamos “¿Qué cosas aprendidas en esta materia son las que vamos a usar para que funcione? Estamos viéndolas por una razón ¿no?”. Al inicio las únicas ideas que sabíamos eran: 1.- tendremos que separar las funciones en un .h y .c y que se trataría de una matriz.

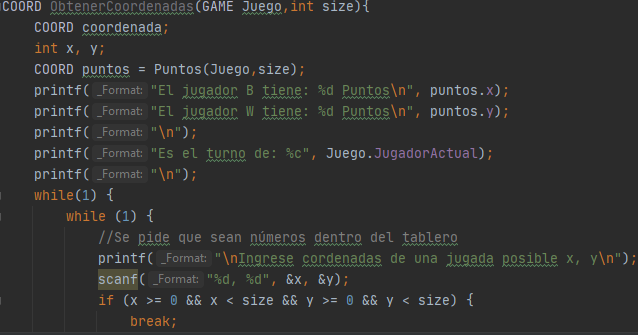
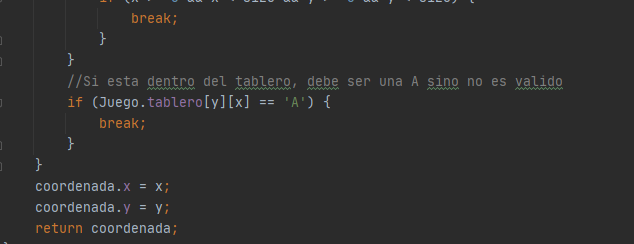
Comenzamos el desarrollo sabiendo que además del tablero hay otro dato que necesitaríamos conocer, que es el turno. Por lo que creamos una estructura, inicialmente de 8x8 para hacer las pruebas de funcionamiento.

Con esto decidimos iniciar un tablero de 8x8 y colocar las fichas iniciales en las 4 casillas centrales.

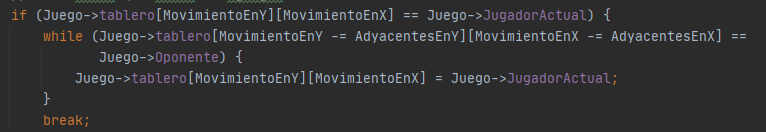
Ahora que se tenía la matriz fue cosa sencilla mostrarla en pantalla con un formato en la consola.

Los desafíos vinieron cuando había que realizar movimientos, podía ingresar una coordenada cualquiera, pero debía ser válida para que funcionara. ¿Cómo determino si es válida la casilla ingresada? La idea original era recorrer el arreglo, y encontrar las fichas “aliadas” y a partir de estas comprobar si tenían jugadas adyacentes o cercanas posibles.

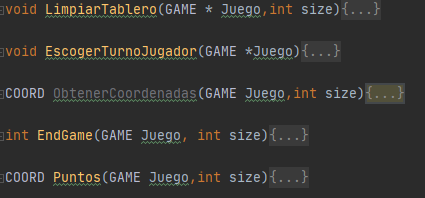
De esta manera, Terminamos viendo esto bastante más complicado de lo que esperábamos, así que en su lugar examinamos todas las casillas vacías, en búsqueda de ver si es una jugada posible para el jugador actual (busca si cumple las condiciones de encerrar a una o varias fichas opuestas entre su color).

Una vez hecho esto de distinguir casillas posibles, teníamos que realizar los movimientos, originalmente la función era demasiado compleja, incluía user input, y bastantes recorridos de ciclos para poder funcionar correctamente, el user input dentro de esta misma función, trabajaba bien en un inicio, pero conforme crecía y creía, por recomendación la segmentamos en otra parte distinta (Esto mientras usábamos la consola).

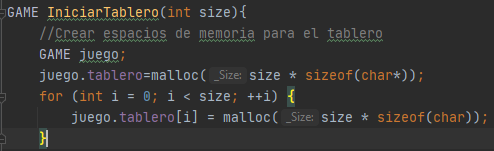
Un cambio de diseño al momento de implementar aquí fue decidir como cambiaria el color de las fichas, si detectaba la dirección y se desplazaba hasta el final o si desde el final iba en reversa convirtiendo hasta el inicio. Terminamos optando por regresar de la ficha recién puesta hasta la ficha original.



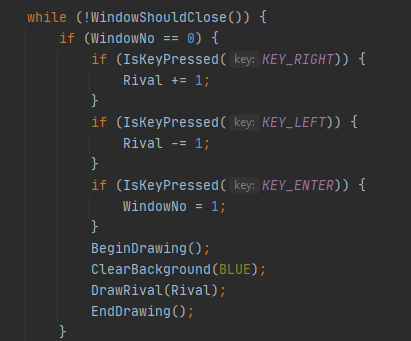
Adicional a estas funciones principales agregamos otras que nos ayudarían a limpiar el tablero antes de hacer otra jugada posible, otra que nos ayudara con el marcador, una que nos ayudase a determinar quién era el jugador en turno y por último una que nos avisara cuando el juego debía detenerse.

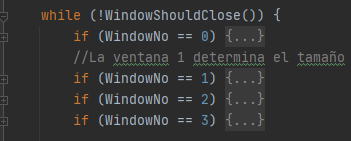


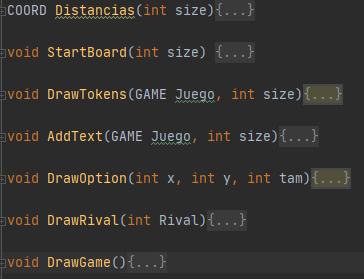
Con todo esto listo reversi ya estaba hecho, nuestro problema era que necesitábamos un infinireversi, Infinireversi implicaba un tablero de tamaño indicado por el jugador, aquí llego el uso de otro concepto que vimos en el curso, la poderosa memoria dinámica.

Fueron muchos conflictos los que hubo con la memoria dinámica sobre todo como implementarla. Teníamos problemas “¿Y si el size nos altera todo?¡ Y si no almacena bien?” todo se solucionó mucho más sencillo de lo que parecía

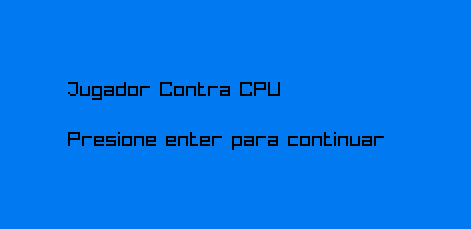
Con esto listo Infinireversi estaba completado, lamentablemente solo en consola lo siguiente es el reto Raylib. Este fue todo un reto a pesar de que ambos nos encontrábamos mínimamente familiarizados con una interfaz gráfica. El mayor esfuerzo fue comprender como se comportaba la librería y de ahí lo sencillo fue adaptar partes del juego a una representación visual.

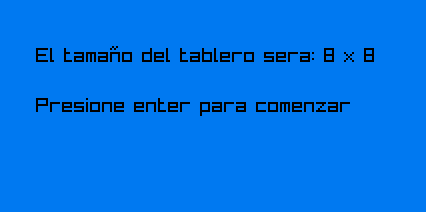
Inicialmente supimos hacer pantallas estáticas, no se actualizaban, solo se creaban y se mostraban. Debemos reconocer que fue todo un problema intentar añadir las funciones que tuvieran user input porque nos generaba un conflicto al compilar dejando sin responder la pantalla creada, terminando la ejecución o simplemente sin respuesta. Al darnos cuenta de esa complicación decidimos cambiar la forma en que el usuario metía datos al juego, paso de entrada en la consola, a manipulación con el teclado. El teclado facilito muchas cosas como el movimiento y la realización de movimientos.

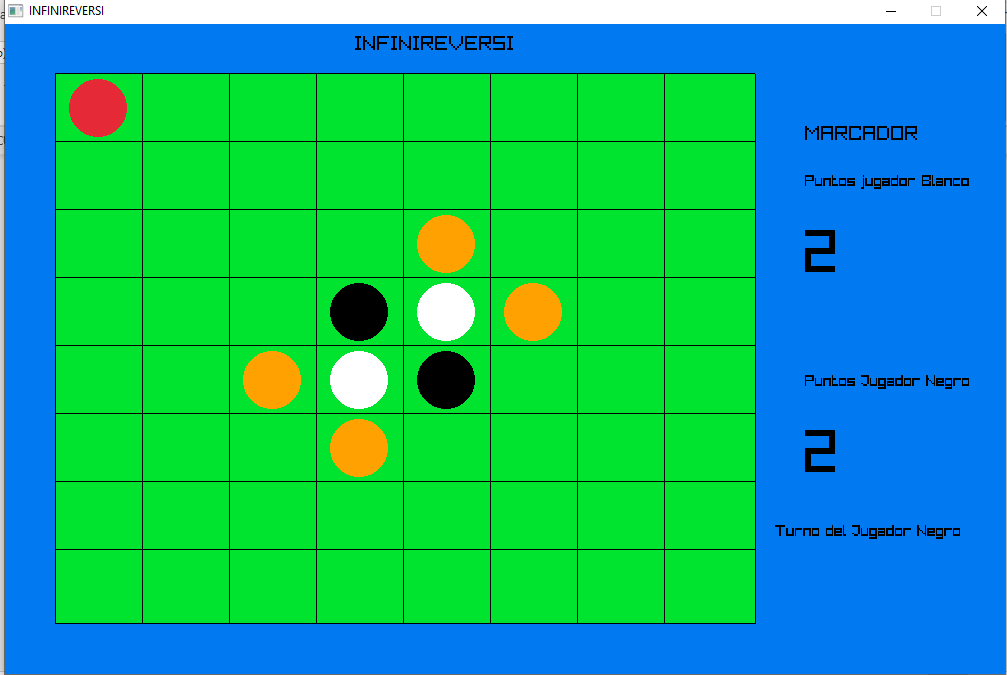
Así logramos implementar Reversi en Raylib, lamentablemente infinireversi era distinto, el usuario originalmente debía meter los datos, pero no teníamos esa opción, inténtanos encapsular pantallas en funciones y que retornaran valores como el tamaño del tablero, el movimiento en el juego etc. Ahí nos dimos cuenta de que Raylib solo permite crear una pantalla por ejecución, entonces los planes de varias funciones que crearan pantallas distintas pasaron a una función que dibujara diferentes pantallas dependiendo de si una es válida o no. Al terminar de usar una pantalla había que cambiar a que pantalla nos moveríamos y por eso indicamos a que pantalla iríamos modificando un valor que llamamos Numero de pantalla, decidíamos que acciones se tomaban y que se dibujaría en pantalla.

Raylib consistió en especificar que queríamos escribir en pantalla y que teclas como entrada de datos íbamos a usar. Creamos distintas funciones para determinar distancias a las que estarían las líneas del tablero, el tamaño de las fichas, escribir un marcador, donde se dibujarían las fichas y un apoyo visual que es la posición del jugador.

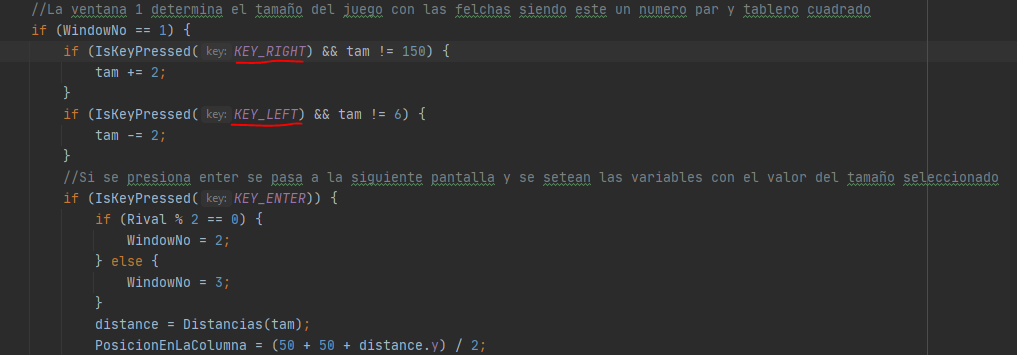
El resultado final fueron 4 pantallas, una que le indica al usuario su rival, otra que especifica el tamaño y 2 que se refieren a una ia que a continuación explicaremos y otra que es el modo competitivo del juego, enfrentándote a la persona que tengas a un lado.







La manera en que logramos que el usuario escogiera contra quien jugar y el tamaño del tablero fue que en lugar de que el mismo usuario ingresara los datos, solo con las flechas de izquierda y derecha pudiera cambiar en las opciones y al final solo diera enter como dice en las pantallas, y funcionó bien.

Una vez logrado esto, comenzamos a pensar, la pantalla de jugador con jugador funciona de manera sencilla, solo se va corriendo un ciclo mientras el juego no termine en el cual pide posición, calcula puntuación e indica de quien es el turno.

En cuanto a la IA fue un poco mas complicado. Después de haber investigado varios algoritmos que se nos indicaron como posibles a usar, optamos por utilizar el MINIMAX.

El algoritmo de MINIMAX funciona de la siguiente manera:

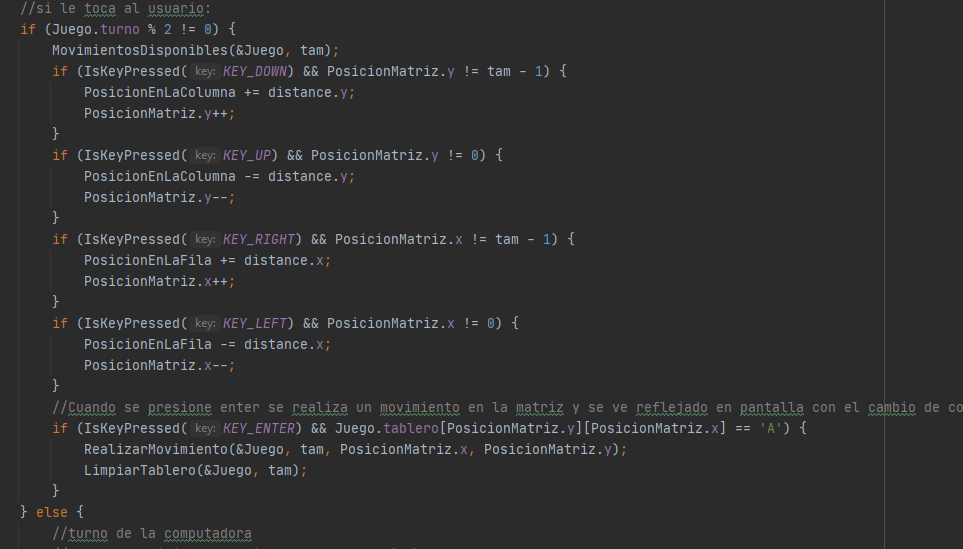
Tenemos una estructura, un árbol de nodos y en cada nodo se encuentra un estado del tablero que es posible tomar desde un nodo padre el cual es el estado actual del tablero

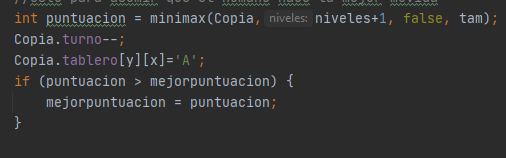
Cada nivel son los movimientos disponibles de el siguiente jugador, que pueden realizarse del tablero anterior. Acomodar los datos de esta manera permite a la IA tener una visión a futuro de los movimientos que se realizarán después. Lo que sigue es de alguna manera, encontrar la forma de darle un valor a cada estado del tablero, y así, dependiendo del jugador, escoger entre el movimiento que maximice o minimice el valor. De aquí el nombre de MINIMAX.

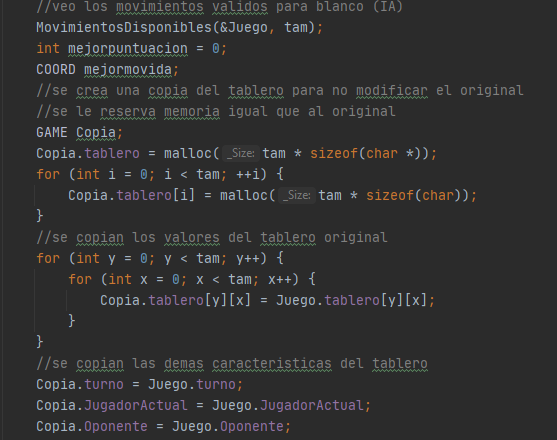
Después de haber comprendido la teoría de minimax entonces surgió lo difícil o complicado, pues todos los ejemplos que había en internet del algoritmo eran resolviendo el juego de gato, entonces, separamos las partes importantes del algoritmo:

1. Un valor para cada tablero
2. Los movimientos siguientes
3. Una condición para que deje de evaluar

Entonces pensamos en que, para poder adaptarlo a nuestro juego debíamos usar funciones que ya teníamos. Entonces, En la ventana que creamos para la IA, se puso una condición, si es el turno de los negros, entonces el juego pide los valores de manera normal, si no, era turno de la IA y fue ahí donde empezamos el código.

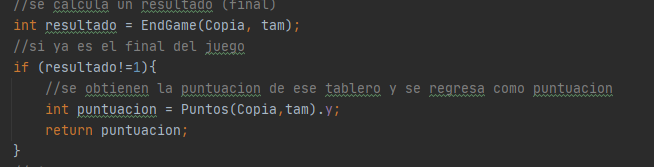


Para el valor que tendría cada nodo decidimos usar la función del marcador, según el nivel en que entraba la IA, el score se calculaba para ese tablero y ese era el valor que se consideraba, al final, el blanco busca tener el mayor numero de fichas y el negro igual, entonces si hacíamos que se escogieran los tableros en que el score era mayor para ese jugador, ya teníamos esa parte resuelta. 

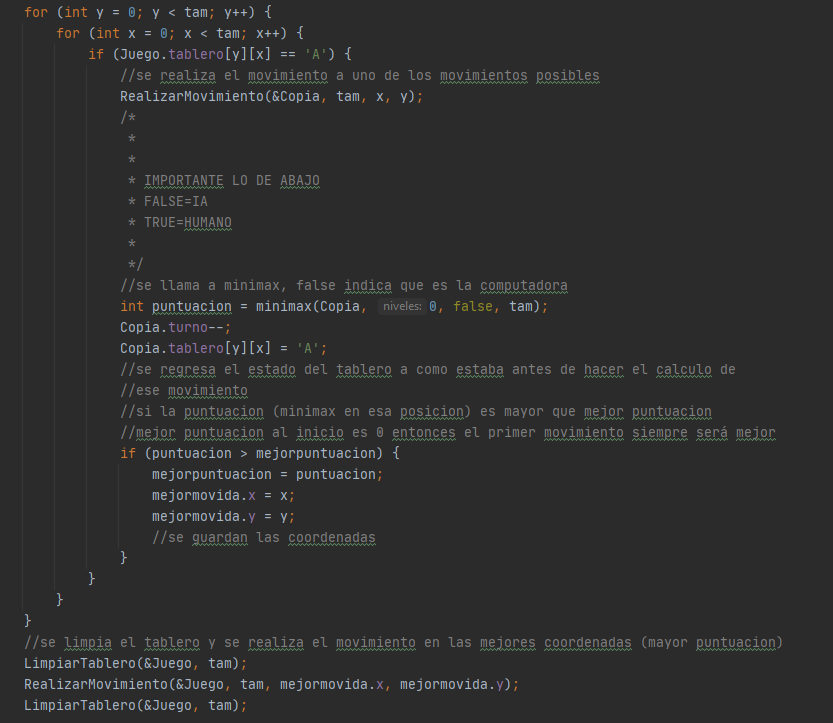
En la parte de movimientos siguientes tomamos la función de Movimientos disponibles, pero como esta función modifica el tablero agregando dichos movimientos, entonces creamos una copia del tablero igual de manera dinámica sobre la que trabaja la IA.

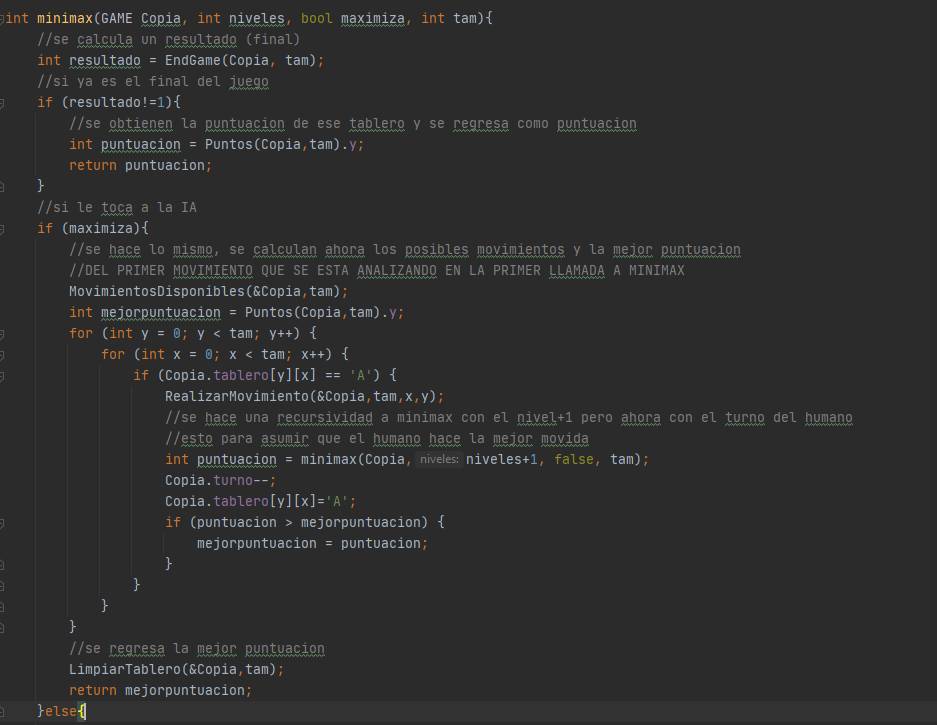
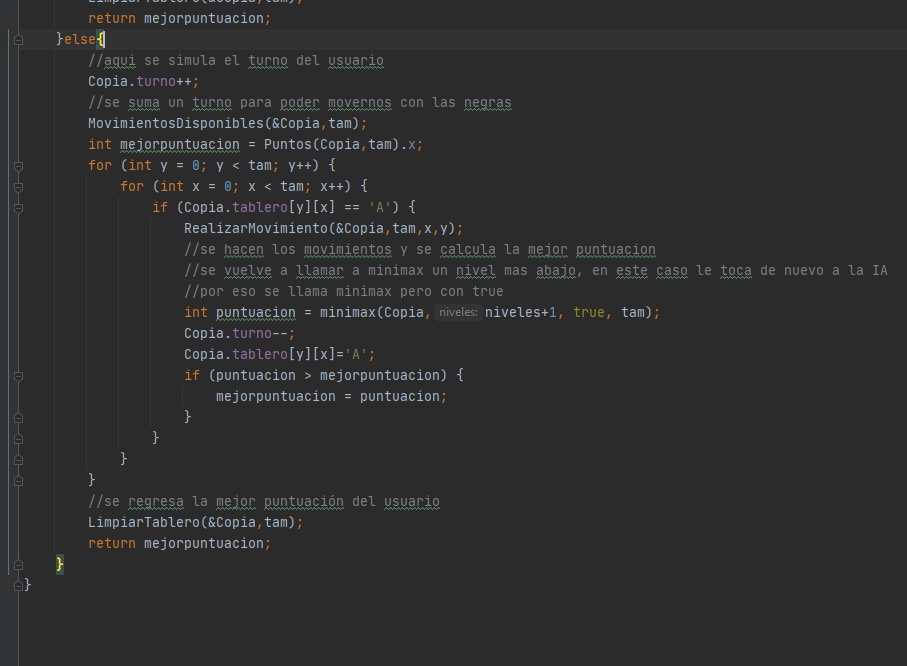
Se calculan los movimientos disponibles y después se recorre el tablero en busca de una “A” o movimiento disponible y si lo encuentra, se escoge ese movimiento y vuelve a hacer lo mismo, se llama a Movimientos disponibles, de ese movimiento y sucesivamente hasta terminar el juego.

Para la condición de termino, como ya dijimos se escogió que se realice hasta que la función de End Game sea verdadera.



Una vez que se realizaba la función minimax en ese movimiento que encontramos, entonces, se guardaba el valor del score mayor y las coordenadas acordes a él. Se regresa el estado del tablero a como estaba antes de escoger ese movimiento, y entonces se busca otro posible, y se vuelve a hacer lo mismo. Al final, como ya teníamos cual era el mejor movimiento, solo se realizaba, pero en el tablero oficial.



Función de minimax:

Con esto resuelto, y el código adaptado a nuestro juego, la IA al momento de tener su turno ya podía evaluar los siguientes movimientos y escoger cual sería el que mejor le fuera.

Al final solo pulimos detalles de algunos bugs que aparecían pero en general, el proyecto ya estaba terminado, logramos construir un reversi con tablero variable que se almacena con memoria dinámica, con una interfaz gráfica y la opción de escoger jugar contra la computadora.