**Pregunta C1. Explique el código negamax-a-b entregado:**

1. **Utilice ejemplos de evaluaciones que ilustren para qué sirve cada función.**

Para la realización de este apartado nos limitamos a mencionar el ejemplo adjunto al código de la entrega del negamax con poda. En este ejemplo se mira tanto el buen funcionamiento (comparando la salida del negamax proporcionado con nuestra implementación con poda) como una bajada, que resulta ser más que sustancial, en tiempo de ejecución y espacio de memoria utilizado.

**b. Comente línea a línea las funciones implicadas en la implementación del algoritmo negamax.**

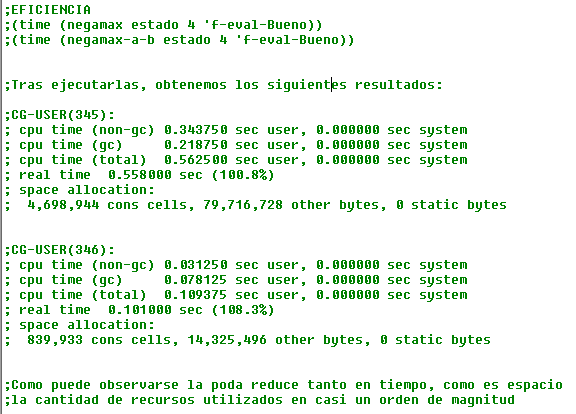
De la misma manera, los comentarios sobre el funcionamiento de código se entregan sobre el script.

A rasgos generales, destacamos la implementación de dos funciones, una de ellas para inicialización y otra para realizar la recursión (siguiendo el ejemplo del negamax sin poda).

Esta segunda función tiene como parámetros extra los límites del intervalo alpha-beta. La clave para la realización de la poda es comparar estos dos parámetros y comprobar la condición de parada (el intervalo resultante de una iteración está mal formado o tiene longitud 0).

**Pregunta C2. Compare el tiempo que tarda un jugador utilizando negamax y utilizando negamax con poda alfa-beta. Comente los resultados.**

Esta es una de las cuestiones que se tratan en el ejemplo entregado junto a la implementación del negamax. Sin embargo, para facilitar la tarea se adjuntan los resultados a continuación.

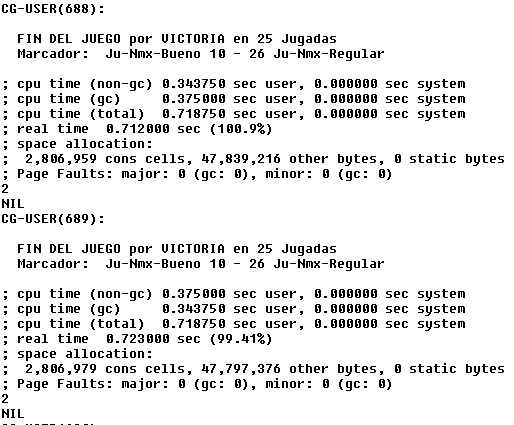


**Pregunta C3. Modifique el orden el que se exploran las jugadas. Comente el efecto que tiene en la poda alfa-beta modificar dicho orden.**

Hemos optado por modificar el orden haciendo simplemente un reverse de la lista de estados. La diferencia entre una implementación y otra es mínima. Hemos comprobado que en ambas el resultado obtenido por el algoritmo negamax es el mismo, lo cual es correcto, pues el movimiento no debe depender del orden de exploración a no ser que dos estados tengan el mismo valor asignado por la función de evaluación.

Se ha comprobado también el tiempo de ejecución y el espacio empleado por el algoritmo, la diferencia es insignificante. Esta diferencia se debe a que el orden de exploración de los nodos sí que influye en la poda, y con ello tanto en el tiempo como la memoria requerida a lo largo del algoritmo.

Todo esto puede observarse en la siguiente captura:



La primera ejecución se corresponde a una partida entre los dos jugadores proporciondos, Bueno y Regular.

La segunda ejecución es una partida entre estos mismos jugadores en un juego con negamax-reverse.

Como peude observarse ocurren todos los fenómenos comentados anteriormente.

**Pregunta C4. Explique los fundamentos, el razonamiento seguido, la bibliografía utilizada y las pruebas realizadas para la entrega de las funciones de evaluación entregadas en el torneo.**

Para la creación de funciones de evaluación no hemos usado ninguna documentación, nuestra principal guía ha sido la practica e intuición (Esta vez la inspiración nos encontró trabajando), si bien es cierto que nos han servido de ayuda varios links encontrados tras sucesivas búsquedas que van desde lo más particular (heurísticas bot mancala, como ganar al mancala…) a lo más general (como decidir una buena heurística para mi IA…). Cabe destarcar un pdf publicado por la universidad de Kansas.

<https://fiasco.ittc.ku.edu/publications/documents/Gifford_ITTC-FY2009-TR-03050-03.pdf>

* Durante los primeros días nos dedicamos en su mayor parte a aprender los fundamentos del juego, entender la API proporcionada para el desarrollo y buscar heurísticas que funcionasen bien contra los pocos bots activos esos días.
* No tardamos mucho en entender que las posiciones más perjudiciales para el juego eran las más cercanas a nuestro kahala. Esto nos llevó a la estrategia de ponderar los hoyos. Esta estrategia funciona relativamente bien, sin embargo, por si misma no es suficiente para mantenerse en el top de la clasificación.
* Cuando nos percatamos de esto decidimos buscar nuevas ideas, nos dimos cuenta que la forma de ganar era conseguir tener fichas en nuestro kahala, por ello implementamos una nueva sentencia que puntuaba con un gran peso que hubiese una gran cantidad de fichas en nuestro kahala. Esto fue beneficioso, una vez implementada esta mejora conseguimos superar al jugador que estaba número 1 en ese momento, el jugador “dos”.
* Posteriormente pasamos por otro periodo donde solo podíamos mejorar el bot que teníamos, debido a la falta de ideas, pero se nos ocurrió buscar la aparición de ceros en nuestro tablero para poder **robar** fichas, lo cual combinado con que nuestro bot se deshacía de las fichas mas cercanas al kahala(cubriendo así la posibilidad de que el enemigo tuviese ceros) resulto ser una mejora sustancial(si bien es cierto que no se produce este efecto deseado el 100% de las veces).
* A partir de este momento llegamos a un atasco, no éramos capaces de capaces de crear un bot que ganase a los ya implementados y probados. Fue por esto que decidimos solicitarte ayuda. Nos diste varias ideas nuevas, algunas de ellas no nos funcionaron (calcular distancias), otras si resultaron fructíferas (ponderar de manera distinta al rival, eliminar asimetrías). Pero esto no supuso una mejora sustancial, dado que llegaron nuevos bots a los que a pesar de los esfuerzos no nos ha sido posible superar. Teníamos una gran esperanza en los algoritmos genéticos, sin embargo, no conseguimos crear una buena y variada primera generación, lo que causo que al final llegásemos a la misma situación de la que partíamos.

**Pregunta C5. El jugador llamado bueno sólo se distingue del regular en que expande un nivel más. ¿Por qué motivo no es capaz de ganar al regular? Sugiera y pruebe alguna solución que remedie este problema**.