Instituto Tecnológico de Costa Rica

Unidad de Computación

Osvaldo Francisco Barrantes Paniagua

Gabriel Rodríguez Chávez

Sede San Carlos

15 Mayo 2016

Tabla de contenido

[Introducción 3](#_Toc451083628)

[Solución del problema 4](#_Toc451083629)

[Análisis de resultados 5](#_Toc451083630)

[Conclusiones 5](#_Toc451083631)

[Recomendaciones 6](#_Toc451083632)

[Referencias 6](#_Toc451083633)

# Introducción

El presente proyecto pretende que se indague sobre diferentes técnicas o algoritmos de árboles de juegos los cuales puedan ser utilizados para sugerir movimientos para el juego de Ajedrez. Para esto también es necesario indagar sobre las reglas básicas del juego, así se tender el mismo y el contexto sobre el cual va dirigido el proyecto. Además, es necesario indagar sobre los distintos lenguajes de programación que permiten la programación multi-core o procesos en paralelo con el fin de mejorar los tiempos de respuesta de los algoritmos. A partir de los algoritmos y lenguajes analizados se escogen los más indicados para el desarrollo de la solución programada.

# Solución del problema

Inicialmente se realiza una investigación sobre los lenguajes conocidos para la implementación de programación con paralelismo y los mejores que existen, de los cuales se concluye que C/C++ usando la librería CUDA, Python y Java son las mejores opciones (otras opciones son PHP, Javascript, Perl, TSQL, Ruby). Tomando en cuenta que C/C++ utiliza una librería externa (y que el proyecto anterior para el curso fue en C#) y que Python no es el lenguaje de mayor interés, se decide que este proyecto se implementa en el lenguaje Java.

Posteriormente corresponde a investigar un poco sobre el juego sin embargo los desarrolladores ya conocen las reglas básicas por lo que se procede a investigar sobre los algoritmos que se pueden emplear para la solución del juego por medio de árboles. La mayor cantidad de información habla sobre los algoritmos Maximax, Maximin, Minimax, Minimax con poda, todos ellos comparten el siguiente contexto y funcionalidad:

A partir del estado actual del tablero, se obtienen todos los posibles movimientos válidos a realizar, luego recursivamente para cada movimiento se obtienen los movimientos siguientes hasta que se encuentre en un estado final o que alcance la profundidad establecida por el programador. A cada movimiento se le asigna un valor el cual será el factor que determine cuál es el mejor movimiento. Parte del cálculo del valor se debe a que cada tipo pieza tiene un valor ya determinado.

* Maximax: este algoritmo solo tiene en consideración los movimientos del jugador al que le corresponde realizar el movimiento y sugerirá solamente el mejor de ellos, es decir que para este contexto considerará únicamente la jugada que le otorgue más puntos según las fichas. Un ejemplo de jugada sería que si con la Reina puede obtener X puntos al eliminar una Torre lo hará, sin importarle si el oponente puede eliminar su Reina con un peón, pues el algoritmo esperará que esto no suceda ya que es extremadamente optimista.
* Maximin: el problema con este algoritmo es que solo aplica para juegos de un solo jugador, sin embargo, igualmente se posible implementarlo, a diferencia con el Maximax, este algoritmo es sumamente pesimista y siempre esperará el peor resultado posible por lo que las jugadas que realizaría serían aquellas en donde obtenga un resultado positivo, pero con la menor puntuación.
* Minimax: durante la investigación de algoritmos en general para juegos y luego específicamente para Ajedrez, mucha información validaba Minimax como el mejor algoritmo en este ámbito. De todos los posibles movimientos y estados del tablero, este algoritmo buscará todos los mejores resultados y de todos ellos elegirá el que tenga menos riesgos, buscando así el más seguro y eficaz al mismo tiempo, y es esto lo que lo hace el mejor algoritmo para esta solución.
* Miminax con poda: es el algoritmo Minimax pero para no realizar más cálculos de la cuenta, evalúa si el movimiento que está analizando más bien se está dirigiendo a una peor solución entonces deja de recorrerlo.

Analizando los algoritmos se puede apreciar que en su programación son todos sumamente similares, y lo que cambia en cada uno de ellos es la decisión de cuál es la mejor jugada en su propio contexto. Al ser Minimax el mejor algoritmo estudiado se plantea implementarlo junto con el Minimax con poda, el peso para la decisión de este último algoritmo era conocer que tanto beneficia o perjudica la poda al algoritmo original.

# Análisis de resultados

Según lo analizado para la realización de este proyecto se ha concluido que todas las tareas han sido finalizadas satisfactoriamente. Ver la Tabla 1 para visualizar el detalle de las tareas para este proyecto.

**Tabla 1**

|  |  |  |
| --- | --- | --- |
| Tarea | % Completo | Comentarios |
| Investigación: Ajedrez. | 100% | Sin mayor contratiempo ya que los miembros ya conocen sobre el juego. |
| Investigación: Árboles de juego | 100% | Además del material en Internet también se tomó en cuenta material visto en el curso. |
| Investigación: Lenguajes de programación multicore | 100% | Se encontraron varias fuentes muy útiles, la de mayor peso fue el paper “A comparative study of programming languages in Rosetta code.” (Nanz & Furia, 2015) |
| Investigación sobre algoritmos para el Ajedrez | 100% | La mayor fuente de información encontrada sobre algoritmos de juegos incluido el Ajedrez menciona los algoritmos Minimax, Maximax, Maximin, Minimax con poda. |
| Implementación del juego con ambos algoritmos | 100% | Se implementa un algoritmo para cada jugador. |
| Implementación de paralelismo | 100% | Ambos algoritmos utilizan paralelismo para evaluar los movimientos al mismo tiempo. |
| Sugerencia al jugador | 100% | Ambos algoritmos retribuyen una sugerencia al jugador en turno indicando la fila y columna donde está la pieza y donde debe moverla. |

# Conclusiones

La implementación del proyecto como tal fue satisfactoria, se logró cumplir con los objetivos trazados y de alguna manera obtener una retroalimentación de los temas que se han visto en el curso.

Además, con la investigación sobre los algoritmos de árboles de juegos se reforzó el conocimiento adquirido en el curso de Investigación de Operaciones donde también se estudian los algoritmos Maximax, Minimax, Maximin y algunos otros más.

Indagando sobre los algoritmos que admiten multicore se encontraron algunos lenguajes no esperados y algunos en posiciones de ranking que no se esperaban, por ejemplo, no se conocía sobre los lenguajes como Tcl, Racket, D, Go, tampoco se esperaba que TSQL y F# aparecieran en la lista de los mejores 20 lenguajes para programación multicore.

# Recomendaciones

Finalizado el proyecto se obtuvieron resultados satisfactorios y el proceso de investigación cumplió con su objetivo de conocer un poco más sobre algoritmos y lenguajes de programación multicore característica a la que durante la carrera no se le da mayor seguimiento, además se reforzaron áreas de estudio ya vistas en clases por lo que no se analizaron recomendaciones específicas para el proyecto como tal.

# Referencias

Everything.explained.today. (2016). *Alpha–beta pruning explained*. Obtenido de Alpha–beta pruning explained: http://everything.explained.today/Alpha%E2%80%93beta\_pruning/

Everything.explained.today. (2016). *Minimax Explained*. Obtenido de Minimax Explained: http://everything.explained.today/Minimax/

Janet, C., Lu, S.-I., & Vekhter, D. (1999). *STRATEGIES OF PLAY*. Obtenido de STRATEGIES OF PLAY: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/Minimax.html

Nanz, S., & Furia, C. A. (2015). A comparative study of programming languages in Rosetta code. *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, 778-788. doi:10.1109/ICSE.2015.90

NVIDIA Developer. (2013). *NVIDIA Developer*. Obtenido de NVIDIA Developer: https://developer.nvidia.com/accelerated-computing-training